universität wien

DISSERTATION/DOCTORAL THESIS

Titel der Dissertation/Title of the Doctoral Thesis

„Ein kollaborativer Ansatz zur Verbesserung der Informationsdarstellung im IT-Projektmanagement“

verfasst von/submitted by

Mag. Marcello Presulli

angestrebter akademischer Grad/in partial fulfilment of the requirements for the degree of

Doktor der Wirtschaftswissenschaften
(Dr. rer. oec.)

Wien, 2015/Vienna, 2015

Studienkennzahl lt. Studienblatt/ degree programme code as it appears on the student record sheet:	A784 175
Dissertationsgebiet lt. Studienblatt/ field of study as it appears on the student record sheet:	Wirtschaftsinformatik
Betreut von/Supervisor:	ao. Univ.-Prof. Dipl.-Ing. Dr. Renate Motschnig

Eidesstattliche Erklärung

Ich, Mag. Marcello Presulli, erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken wurden als solche kenntlich gemacht.

Ferner wurde nach bestem Wissen und Gewissen gehandelt um etwaige Urheberrechte zu wahren. Sollten wider Erwarten dennoch Fehler unterlaufen sein, möchte ich mich hiermit entschuldigen und werden für die rasche Richtigstellung nach Kenntnisnahme sorgen.

Diese Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Unterschrift Wien, 2015

Abstract

The objective of this thesis initially evolved from an observation in practice. The goal was to make a contribution in form of a problem reduction using a software tool, due to the increasing complexity in IT-Project management. For this purpose, a novel model was developed, which should cover the needs of technicians and management alike. The aim was to develop a better and more transparent understanding of the project. To connect these two domains the plan was the determination of a consensual request of minimality through collaborative survey technique. For this purpose, industrial partners were selected that showed a high degree of heterogeneity to further encourage the discourse in the collaborative sessions. Based on the results, a specification has been derived, which is reflected in its implementation as a software prototype. Guided by the original motivation of the project hypotheses have been postulated, which need to be verified in a subsequent testing phase. In this test phase, specified with the companies in advance, the core functionality was tested in practice. The results were captured by questionnaires, interviews and general observations from field research in order to be able to make conclusions about corporate benefits. The outcome gave a further direction with regard to a possible rejection or deepening of the applied concepts and realization.

Keywords: IT-Projectmanagement, Collaboration, VCS, RDF/XML, SVG, DSL, Scala

Kurzfassung

Die Zielvorstellung dieses Werkes entwickelte sich anfänglich aus einer Beobachtung in der Praxis. Es wurde hierbei versucht, bedingt durch die zunehmende Komplexität im IT-Projektmanagement, einen Beitrag zur Problemminderung mittels eines Software-Werkzeuges zu leisten. Dazu wurde ein neuartiges Modell entwickelt, welches die Anforderungen der Techniker und des Management gleichermassen erfassen soll, um ein besseres und transparenteres Projektverständnis zu entwickeln. Um diese zwei Domänen zu verbinden bedient sich das Vorhaben mittels einer kollaborativen Erhebungstechnik um die konsensualen Minimalanforderungen zu ermitteln. Es wurden dabei Industriepartner gewählt, die eine hohe Heterogenität aufwiesen um dadurch in den kollaborativen Sitzungen den Diskurs weiter anzuregen. Basierend auf den Ergebnissen, wurde daraus eine Spezifikation abgeleitet, die sich in ihrer Implementierung als Software-Prototyp widerspiegelt. Geleitet von den ursprünglichen Motivationen des Projektes wurden Hypothesen postuliert, die es in einer anschließenden Erprobungsphase zu verifizieren galt. In der mit den Unternehmen spezifizierten Testphase wurde die Kernfunktionalität in der Praxis getestet um mittels Fragebogentechnik, Interviews und generellen Beobachtungen aus der Feldforschung, Schlussfolgerungen bezüglich eines Unternehmensnutzens anstellen zu können. Diese Ergebnisse gaben dann die weitere Richtung im Hinblick auf ein mögliches Verwerfen oder Vertiefen der hier angewandten Konzepte und Realisation vor.

Vorwort

Die Entstehung dieser Arbeit basiert vor allem auf den praktischen Erfahrungen die ich vor einigen Jahren machen durfte. Es begann im Sommer 2008, ich war ein junger Entwicklungsleiter eines kleinen mittelständischen Unternehmens, das sich der Entwicklung und Erzeugung telekommunikationsspezifischer Einrichtungen widmete. Die Entwicklungsstandorte waren auf Österreich und Deutschland verteilt. Zu dieser Zeit arbeiteten wir mit Hochdruck an einem neuartigen kommunikationsplattformübergreifenden Gerät.

An einem heißen Julitag in jenem Jahr stand wieder ein zyklisches Projektstatusmeeting auf der Agenda. Als die Besprechung voll im Gange war, und die üblichen aggregierten Kennzahlen präsentiert wurden, erkannte man während des Vortrages wie die Moderatoren zunehmend unsicherer wurden. Nach einigen detaillierteren Nachfragen wurde klar, dass jene Zahlen auf einer sehr unzuverlässigen Basis standen, da in diesem Zeitraum nämlich einige Wissensträger und Kompetenzfaktoren im wohlverdienten Urlaub , beziehungsweise auch krankheitsbedingt abwesend waren.

Mir wurde schmerzlich bewusst wie „labil“ und „empfindlich“ der individuelle Kommunikationspfad innerhalb einer Projekthierarchie im Eigentlichen war – es gab offensichtlich keinen geeigneten „Backup Pfad“ zur Überbrückung solcher Informationsengpässe, eine semantische Kluft riss in diesem Projektkontext auf. Letztendlich wurde das Meeting unbefriedigend zu Ende geführt und einige, darunter auch ich, verließen den Raum mit Unbehagen.

Getrieben von dem Verlangen Abhilfe zu leisten, reifte nach einiger Zeit folgende Idee: *„Es muss doch wohl möglich sein eine einfache Konvention zu entwickeln, die Techniker und Manager gemeinsam tragen, und mittels eines Werkzeuges einige der immer wiederkehrenden Informationsartefakte und Metriken, die von Interesse sind, nach Bedarf zu ermitteln.“*

Bestärkt von diesem Vorhaben führte ich mit einigen Kollegen und anderen befreundeten Unternehmen bezüglich Durchführbarkeit und Umsetzung regen Gedankenaustausch. Meine erste Vorstellung war eine „domänenspezifische Sprache“ zu entwickeln die gleichermaßen firmenintern verstanden wird, und diese auf eine essentiell reduzierte Semantik überführt. Es zeigte sich bereits nach einigen Diskussionen, dass es gewiss Bedarfsträger gibt, jedoch basierend auch einfacheren Doktrinen. Die vorerst vage Vorstellung wurde zu einem immer konkreter werdenden Projekt, das letztendlich in einer Software-Tool Implementierung namens *„Repository Based Management Support Tool“* endete.

Im weiteren Verlauf dieser Arbeit werde ich nun die Konstruktion dieser These im wissenschaftlichen Kontext behandeln und eine mögliche Implementierung dazu vorstellen, die auch einer praktischen Erprobung unterzogen wird.

Marcello Presulli Wien, 2015

Danksagungen

Mein Dank gilt allen die mich in meiner Anstrengung einerseits beratend andererseits bestärkend unterstützt haben. Meiner Familie die mich immer so gewähren ließ wie ich es für nötig hielt. Auch allen Industriepartnern die vor allem in der Entwurfs und Erprobungsphase mich begleiteten und immer wieder die praktische Seite meiner Arbeit aufzeigten.

Einen speziellen Dank schulde ich meiner Doktormutter Prof. Dr. Renate Motschnig die mir so viel Geduld entgegenbrachte und in ihren Seminaren stets weitere interessante Aspekte motivierte, vorallem im Rahmen des EU-weiten iCom-Projektes, welches für mich durch die Kompetenz der Teilnehmer verschiedenster Disziplinen eine wahre Bereicherung darstellte.

Ich möchte mich auch allgemein bei all den unzähligen tatkräftigen *Communities* der O*pen Source* Bewegung für die Bereitstellung der teilweise wunderbaren Softwareprodukte, die ansonsten ohne einen gewissen Enthusiasmus nie zustande gekommen wären, bedanken. Auch die Professionalität und rasche Antwortzeit mit der die meisten Anfragen beantwortet werden, hinkt einem kommerziellen Support in nichts nach.

Schriftauszeichnung

Die vorliegende Dissertationsschrift wurde mit dem Desktop-Publishingsystem *PagePlus X8* unter Microsoft Windows 8.1 erstellt. Um den Text nicht zu stark zu überladen, werden ausschließlich die folgenden Auszeichnungen zur syntaktischen Hervorhebung genutzt:

Schriftart: MS-Windows OpenType Systemschrift Constantia, 12pt
Zeilenausrichtung: Blocksatz bei erster Absatzzeile einen Tabstopp eingerückt
Zeilenabstand: 1,2
Akronyme und Firmennamen: VERSAL
Eigennamen, Juxtapositionen und Zitate : *Kursiv* und in „Anführungszeichen“
Quellcodeausschnitte: *Kursiv*

Das Ausgabeformat PDF ist grundsätzlich für die Bildschirmdarstellung optimiert. Sämtliche Abbildungen wurden aufgrund der Dokumentengröße komprimiert. Zwecks vereinfachter Navigation wurde auch der PDF-Lesezeichenindex an die tatsächliche Seitenanzahl angepasst.

Inhaltsverzeichnis

DISSERTATION i

Eidesstattliche Erklärung ii

Abstract iv

Kurzfassung iv

Vorwort v

Danksagungen vi

Schriftauszeichnung vi

Inhaltsverzeichnis vii

1.Einführung und wissenschaftliche Motivation 11

1.1 Allgemeines 11

1.2 Einleitung 11

1.3 Motivation und Hypothesen 12

1.4 Forschungsmethodologie 15

1.5 Die Kernthese als Forschungsfrage 17

1.6 Gliederung 19

Teil I 21

2.Historischer Überblick und vergleichende Studien 22

2.1 Hintergrund 22

2.2 Verwandte Arbeiten 27

Teil II 33

3.Theoretische Betrachtungen und Klassifikationen des Projektmanagements 34

3.1 Das klassische IT-Projektmanagement 34

3.2 **Die Rollen und Mittelverteilung der Projektorganisation** 38

3.3 **Support Systeme der IT** 41

4.**Kommunikationsparadigmen und Einflussfaktoren im Projektablauf** 45

4.1 **Strukturelle Probleme im Kommunikationsablauf** 45

4.2 **Personenzentrierte Kommunikation – ein Ansatz zur verbesserten Projektkollaboration** 47

4.3 **Effekte und Gesetzmässigkeiten im Projektablauf** 47

4.4 **Theoriebezüge zur Thesenbildung** 51

5.**Identifikationen und Forschungskonzeption** 53

5.1 **Projektabgrenzungen** 53

5.2 **Problemidentifikation** 54

5.3 **Das applikative Forschungskonzept** 58

6.**Die These und ihre Anforderungen** 69

6.1 **Repository Based Management Support System** 69

6.2 **Die Notwendigkeit einer eigenen Ontologiesprache** 72

6.3 **Das RBMSS Vokabular** 75

6.4 **Typisierung des Namensraumes von RBMSS** 79

Teil III 85

7.**Die Basisentscheidungen der Implementierung** 86

7.1 **Das Projekthandbuch als Anforderungsbasis** 86

7.2 **Die Wahl des Versionskontrollsystems** 87

7.3 **Die Basismechanismen des Wirtssystems** 88

7.4 **Sinnvolle Konventionen von RBMST** 92

7.5 **Data Mining und Reasoningmechanismus** 93

8.Die Funktionalität des Wirtsystems und die Auswahl einer Entwicklungsplattform 99

8.1 **Der Implementierungssansatz** 99

8.2 **Essentielle Implementierungsentscheidungen** 111

8.3 **Die Festlegung des Entwicklungssystems** 119

9.Die konzeptionelle Architektur des Software Prototypen 123

9.1 **Der Software Prototyp als Forschungssoftware** 123

9.2 **Die Auswahl eines adäquaten Vorgehensmodells** 125

9.3 **Die Software-Architektur** 126

9.4 **Aspekte der Performance und Ergonomie** 144

10.Die prototypische Forschungssoftware RBMST 151

10.1 **Die Produktivversion RBMST** 151

10.2 **Die Benutzerschnittstellen** 153

10.3 **Die Artefaktausgabe** 161

Teil IV 165

11.Ermittlungsmethoden und die Wahl der Industriepartner 166

11.1 **Die Wahl der empirischen Ermittlungsmethode** 166

11.2 **Die Feststellung des Unternehmensnutzen** 168

11.3 **Die Industriepartner** 170

12.Fragestellungen und Manifestationen der Probanden 177

12.1 **Die Evaluationsprozedur** 177

12.2 **Die Strategie der Fragestellungen** 178

12.3 **Die Manifeste** 178

13.**Die Auswertungsverfahren und ihre Ergebnisse** 183

13.1 **Die Auswertung** 183

13.2 **Die Rohdaten** 185

13.3 **Die Hypothesenbewertung** 203

13.4 **Schlussfolgerungen** 206

Teil V 211

14.**Diskussionen und weitere Erörterungen** 212

14.1 **Diskussion** 212

14.2 **Behandlung offener Themen** 214

14.3 **Grenzen der Studie** 219

14.4 **Ausblick** 222

14.5 **Schlusswort** 223

Anhang 225

Glossar und Abkürzungsverzeichnis 233

Quellen- und Literaturverzeichnis 234

Abbildungsverzeichnis 242

Auflistungsverzeichnis 245

Tabellenverzeichnis 246

Curriculum Vitae 247

1 Kapitel

Einführung und wissenschaftliche Motivation

"Any problem in computer science can
be solved with another layer of indirection.
But that usually will create another problem."
David John Wheeler

Schwerpunkte in diesem Kapitel:

- Allgemeines
- Einleitung
- Motivation und Hypothesen
- Forschungsmethodologie
- Forschungsziele
- Gliederung

1.1 Allgemeines

Es werden im Verlauf dieser Abhandlung ausschließlich geschlechtsneutrale Personenbezeichnungen verwendet, da zumeist eine Unterscheidung irrelevant ist. Nur in Ausnahmefälle, wo eine geschlechterspezifische Benennung sich als notwendig erweist, wird davon Abstand genommen und das Geschlecht berücksichtigt.

1.2 Einleitung

Das interdisziplinäre Projektmanagement ist eines der am meist diskutiertesten Themen an sich. Es symbolisiert eine stark heterogene Schnittstelle, die es aus den verschiedensten Domänen dennoch einheitlich zu bedienen gilt. Einerseits gibt es eine Fülle an Werkzeugen und Methoden andererseits jedoch ist deren Anwendung und Nutzen von Projekt zu Projekt meist unterschiedlich. Die individuellen Schwerpunkte, Gegebenheiten und Rahmenbedingungen lassen den Grad der Komplexität oft zu schnell anwachsen. Die Praxis gibt meistens in einem paradigmatischen Trajekt[1] die Antwort darauf [ND2002].

Viele Manager bemühen sich klassischer Tools wie dem Microsoft Excel Paket da sie es einfach so gewohnt sind. Wie auch bereits in der *Planning Survey 2014*

[1] eine Vorgehensweise, die etwas Bestehendes bis zum Exzess ausreizt

Studie[1] angedeutet wurde, versuchen sie mit exzessiven Techniken die jeweiligen Softwarepakete für den angedachten Einsatzzweck zu zweckentfremden, die Resultate sind jedoch häufig suboptimal. Ein komplementäres Problem dazu stellt generell das mangelnde Metawissen, das Wissen um spezialisierte Werkzeuge des Projektmanagements, beziehungsweise auch die Medienkompetenz an sich dar. Viele Leitungshandelnde[2] haben einfach nicht mehr die Zeit und Geduld sich in neue Tools einzuarbeiten, beziehungsweise fehlt manchen Klein und Mittelunternehmen auch der finanzielle Spielraum für die Anschaffung der Werkzeuge und vor allem die Schulung der handelnden Personen. Somit sind einerseits die „Total Costs of Ownership" andererseits das mangelnde Metawissen die Haupttreiber für diesen Teufelskreis.

Eine weitere Problematik ist teilweise das aggregierte Niveau der betrachteten Projektkennzahlen. Viele Unternehmen verwenden stets die gleichen Bewertungsfaktoren ohne sich zuvor im Detail ein Bild über die potenziellen Schwerpunkte des Projektes gemacht zu haben. Dies kann einerseits dazu führen, dass die jeweiligen Indikatoren die tatsächliche Projektsituation zu spät widerspiegeln oder ihre Größe nach einfach falsch gewichtet werden, häufig ist teilweise die Abstraktion, sprich das Aggregationsniveau, nicht adäquat. Durch diese falschen Interpretationen und Einschätzungen kommt es dann oft zu Fehlallokationen oder unüberlegten Akquirierungen beziehungsweise Mitarbeiterentlassungen und spontanen Umbesetzungen innerhalb der gegebenen Projektorganisation.

IT-Projekte stellen ein Management der besonderen Art dar. Durch die zusätzliche Komplexität nehmen auch diesbezüglich die Unsicherheiten fast schon exponentiell zu. Es sind einige Faktoren, wie die erhöhte Interdisziplinarität das abstrakte Produkt Software und auch die Projektkomplexität, die dieses Management so verkomplizieren. Ein wichtiges Moment spielt auch oft die große „semantische Kluft" zwischen den Beteiligten. Die Umsetzenden sind oft zu sehr in Jargons und Details verhaftet um den Managern in den jeweiligen Ebenen die abgegrenzten Sachverhalte klar zu transportieren, aber auch der umgekehrte Fall ist vakant.

Des Weiteren kommt hinzu, dass im Konkreten, Software ein immaterielles Gut darstellt das prinzipiell in *höchstpersönlicher* Form entwickelt wird. Das hat nun weiterhin zur Folge, dass es auch schwer vorstellbar und nachvollziehbar ist, beziehungsweise durch die punktuelle Bearbeitung auf individueller Ebene nun der Kommunikationsfluss bezüglich des Projektmanagement ebenfalls auf wenigen Individuen lastet. Falls nun solche Schlüsselpersonen in ungünstigen Betriebszeiten nicht verfügbar sind, ist das dazu nötige Wissen bereits stark fragmentiert und vage.

Diese zuvor beschriebenen Facetten gaben mir das grundsätzliche Bestreben diesen Themenbereich wissenschaftlich zu beleuchten und ein mögliches Konzept zur Komplexitätsbewältigung und Kommunikationsverbesserung in Form einer Software als Problemminderungsansatz beizutragen.

---

[1] Fuchs Christian (2014): The Planning Survey 2014, BARC-Institut Würzburg

[2] der Leitungshandelnde ist etymologisch gleichzusetzen mit dem Manager

1.3 Motivation und Hypothesen

Aus den zuvor erwähnten Beobachtungen manifestierten sich Motivationen unterschiedlichster Perspektiven. Der wesentlichste Motivationsfaktor ist vor allem die Zielsetzung der Erstellung einer kollaborativ entwickelten Konvention die von allen Projektbeteiligten gleichermaßen getragen wird und deren Einsatz so einfach wie möglich, ohne maßgeblicher Einarbeitungszeit respektive Kosteneinsatz, in Form eines Software Werkzeuges von statten geht. Basierend auf diesen Ausgabeartefakten soll das gemeinsam entwickelte Verständnis angewandt werden.

Kollaboration stellt hier das zentrale Moment dar. Es wird versucht, in Abhaltung mehrerer Sitzungen mit Unternehmensbeteiligten, eine formalisierte Herangehensweise die die Anforderungen an ein besseres Projektverständnis erhebt, durch die Handelnden selbst kreieren zu lassen. Diese Vorgehensweise stellt somit ein reflektierendes Vorhaben dar, welches sich durch Kollaboration der Beteiligten verstärkt und stetig konsistenter wird. Ein weiterer Vorteil der sich dabei einstellt, ist die generelle Anwendbarkeit und auch einfache Wiederverwendbarkeit auf ähnliche Problemklassen. Dieser Prozess stellt prinzipiell, gemäß der Nomenklatur des Prozessmanagements, einen kontinuierlichen Verbesserungsprozesses dar (KVP). Ein Prozess der, wenn er einmal installiert, sich in iterativer Weise, durch die involvierten Expertenteams stets inkrementell verfeinern lässt.

1.3.1 Hypothesen

Aufgrund des zuvor erwähnten Motivationsfaktors ergeben sich diverse Motivierungsansätze die nun in Form bedingter Annahmen zusammengefasst werden. Die hier prägnant formulierten Hypothesen, die jedoch nicht den Stellenwert statistischer Testfälle einnehmen sollen, stellen somit den eigentlichen Motor des wissenschaftlichen Vorhabens dar. Es wird, beginnend ab Kapitel 11, in der praktischen Erprobungsphase der Versuch unternommen, jene in mehrdimensionaler Art & Weise zu beantworten. Es ist die Praxis, die letztendlich die Erwartungen im Sinne einer Hypothesenverifikation unter Beweis stellen wird.

1.3.2 Hypothese – Minderung semantischer Inkompatibilitäten

Es wird erwartet, dass durch die kollaborativ entstandenen Vereinbarungen, eine Konsensbildung auf Basis akkordiertem Verständnis entsteht und so den Projektverlauf im Sinne einer gemeinsamen Betrachtungsweise positiv beeinflusst.

1.3.3 Hypothese – Erhöhte Informationsverfügbarkeit

Durch den Einsatz verteilter Systeme und der pervasiven Verfügbarkeit eines geeigneten Werkzeuges zur Informationsexploration, erhöht sich die bedingte Verfügbarkeit im Verhältnis zur personellen Präsenz signifikant, und durch die automatisierte Informationserhebung wird auch eine erhöhte, da zeitlich entkoppelte, Informationsverfügbarkeit erwartet.

1.3.4 Hypothese – Schnellere Informationsexploration

Es ist plausibel, dass durch den Einsatz moderner Informationstechnologie die Phase der Informationsgewinnung deutlich schneller und zielgerichteter funktioniert als der konventionelle Weg der persönlichen Informationsgewinnung wie zum Beispiel im „Face to Face“ Kommunikationsparadigma. Es kann eine schnelle Ermittlung relevanter Informationsfragmente erwartet werden, falls wie unter 1.3.3 eine erhöhte Verfügbarkeit gegeben ist. Durch die korrespondierende Forschungssoftwareimplementation, die in Kapitel 10 näher erläutert wird, kann somit auch eine effiziente Informationsexploration realisiert werden.

1.3.5 Hypothese – Verbesserte Informationsrepräsentation

Die Aufbereitung der benötigten Informationen ist im klassischen Projektmanagement oft mannigfaltig. Angefangen von Spreadsheets über Mindmaps bis hin zu händischen Notizen auf einem Stück Papier dient alles der Verwendung. Die Generierung eines schematisch konsistenten SVG-Graphen als Repräsentationsinstanz, stärkt durch den formalen Darstellungscharakter, die Erwartung der weiteren Nutzung einer vereinheitlichten formalen Repräsentationsform.

1.3.6 Hypothese – Höhere Informationstransparenz

Da ein wichtiger Teil dieser Information direkt aus den Supportsystemen der Unternehmung bezogen wird, soll auch die Herkunft der Daten integer und für alle qualifizierten Beteiligten einheitlich und nachvollziehbar sein. Durch die Originalität der Informationsquelle und die Möglichkeit der Reproduzierbarkeit, wird von einer höheren Informationstransparenz ausgegangen.

1.3.7 Hypothese – Effizientere Kommunikationspfade

Klassischerweise werden Projekte abhängig von der Art und Umfang in hierarchische Managementebenen unterteilt. Diese Form der Arbeitsteilung hat viele Vorteile jedoch auch den entscheidenden Nachteil einer potenziellen Doppelbesetzung, da die Verfügbarkeit der Kette gewährleistet sein muss. Auch hier, abhängig vom Detailgrad der Information, sollten im Regelfall Erleichterungen augenscheinlich werden, da nicht alle Instanzen der Managementkette permanent interrogiert werden müssen. Unter Voraussetzung pervasiver Informationsverfügbarkeit und gewisser Pragmatiken der Informationsbeschaffung, können sich effizientere Kommunikationspfade in der Informationsermittlung ergeben.

1.3.8 Hypothese – Effektivere Informationssicherheit

Die Datenquelle wovon die Informationen extrahiert werden liegen in fast allen Fällen in Form von Datenbanken vor, die im Regelfall, abgesehen von der referenziellen Integrität, auch zusätzlich unternehmensweit zyklisch gesichert werden. Auch die Übermittlung von Informationen, bedingt durch eine gesicherte Informa-

tionsbasis, können aufgrund der Möglichkeit einer einfachen Verschlüsselung zur erhöhten und integeren Informationssicherheit führen.

1.3.9 Hypothese – Psychosoziale Entlastungseffekte

Zu guter Letzt soll auch der psychologische Stressfaktor der intrinsisch in diversen Meetings innewohnt etwas gemindert werden, da einerseits die Information jederzeit abrufbar und andererseits die Datenbasis jederzeit nachvollziehbar ist und somit ein gezogener Schluss bezüglich einer getroffenen Projektentscheidung transparenter wird.

Die soeben dargestellte Taxonomie bedingt einer technischen Realisation, im Sinne eines Werkzeuges, das für alle Projektbeteiligten zugänglich sein muss. Als Nebeneffekt der benötigten Metadateninjektionen, im Vorfeld terminlicher Firsten, könnte sich durch das Wegfallen von ad hoc Informationsermittlungen eine psychosoziale Entlastung während des Projektablaufes einstellen.

1.4 Forschungsmethodologie

Gemäß den Anforderungen zur kollaborativen Erarbeitung von Konventionen und Vorgehensweisen wurde die „Design Oriented Research“ Methodologie gewählt da sie über den Großteil des Umfanges der Arbeit am Adäquatesten erschien.

„In design-oriented research, the knowledge that comes from studying the designed artifact in use or from the process of bringing the product into being should be seen as the main contribution—the 'result'—while the artifact that has been developed becomes more of a means than an end. Typically, this implies that the artifact that is developed does not need to encompass all services, functions, and level of completeness that a final 'product' would need to embrace. The design-oriented researcher hence works with sketches and prototypes of different kinds, depending on what aspects are Investigated.“[1]

Diese Methodologie forciert den Ansatz der iterativ-inkrementellen Vorgehensweise. Es wird explorativ ein Hypothesendesign modelliert mit der Erwartung, dass die iterative Evaluation eine asymptotische Näherung zwischen Modell und Realität ergibt. Dieser Ansatz ermöglicht letztendlich wissenschaftliche Ergebnisse zu erhalten, im praktischen Kontext zu untersuchen und pragmatische Schlüsse daraus zu ziehen.

Gemäß [YS2009] kann man „Design Oriented Research“ als einen Oberbegriff verstehen, der sich folglich in drei weitere Forschungsrichtungen konkreter untergliedern lässt.

— Formative Research

— Design based Research

— Design and development Research

[1] Daniel Fallman 2009: Why Research-oriented Design Isn't Design-oriented Research

Da bereits während der Anforderungserfassung, Neigungen und Intentionen der Agenten aktiv miteinbezogen werden, um letztendlich eine entwurfgetriebene Forschung anstellen zu können, kommen hauptsächlich die ersten beiden Vorgehensweisen zur Anwendung. *Design and development Research* hingegen weniger, da bei diesem Ansatz verstärkt der Aspekt der konkreten Entwicklungsleistung, sowie die Wechselwirkungen zwischen Produkt und Nutzerschaft die weitere Entwicklung vorantreibt. Im Folgenden wird der Ausdruck *Design oriented Research* hauptsächlich synonym als *Design based Research* verstanden.

Abbildung 1.1 stellt den spiralförmigen Verlauf beginnend mit der Identifikation einer Herausforderung über die Entwicklung eines Prototypen bishin zur Erprobung samt anschließender Bewertung dar. Ausgehend von einem Spezialfall kann durch Reflexion des Prozesses, eine Verallgemeinerung angestrebt werden, welche durch die Generalität und Wiederverwendbarkeit dieses Ansatzes eine Anwendung auf ähnliche Problemstellungen zulässt.

Die Problemanalyse fließt hierbei in das adäquate Vorgehensmodell ein, das durch seinen entwurfsbasierten Anspruch die Entwicklung eines Software-Prototypen forciert und durch die anschließende Evaluation der weiteren Spezifikationsverfeinerung dient. Jede Iterationsstufe wird abschließend, wie auch unter 5.3 dargestellt, durch das Erfassen der gesammelten Daten samt unternehmerischer Nutzenauswertung mit weiterführender Konzeptvertiefung beendet, und einer weiteren Iteration zugeführt.

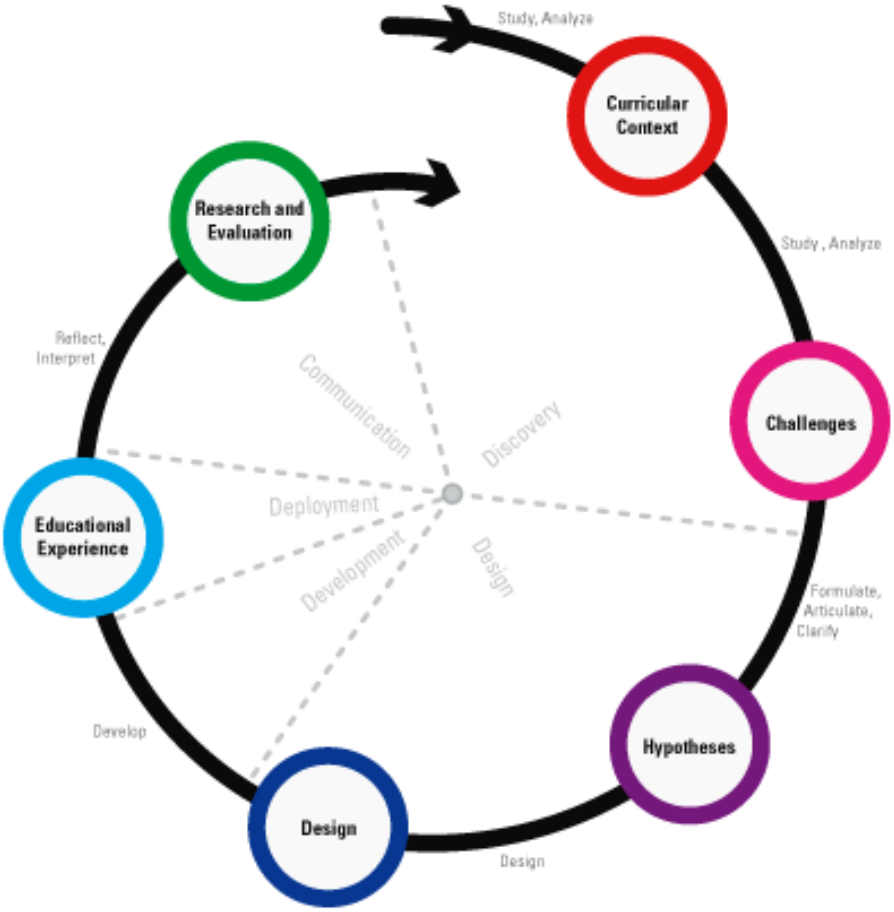


Abbildung 1.1 – Design Oriented Research

1.5 Die Kernthese als Forschungsfrage

Prinzipiell geht die Arbeit der Frage nach, ob eine durch Kollaboration definierte, abgegrenzte „*domänenspezifische Sprache*“ signifikante Einsparungen bzw. Erleichterungen im Bereich der Organisation und Kommunikation in Software-Projekten bringen kann und somit auch gegebenenfalls ökonomische Effekte erzielbar sind. Diese Dissertationsarbeit besitzt aufgrund, inhärenter Managementaufgaben und betriebswirtschaftlicher Ressourcenallokationen, interdisziplinäre Eigenschaften, der Kern jedoch ist aus informatischer Sicht die Bereitstellung einer möglichen Methodik samt Implementierungssansatzes zur Erarbeitung einer Lösung.

Die Praxis ist hier ein Generator, jene relevante Fragestellung in einem wissenschaftlichen Kontext zu erörtern: *„Ein verteiltes Konfigurationsmanagement, das bereits inhärente projektbezogene Kerninformationen besitzt, und diese im Sinne des Projektmanagements als Entscheidungsgrundlage zu nutzen, steht im Zentrum der Betrachtung mit dem Fokus kommunikative wie auch organisatorische Optimierungen zu erzielen.“*

Aufgrund dieser Thesen ergeben sich nun einige charakteristische Fragen die es zu klären gilt:

— *Wie kann ein Werkzeug zur effektiveren Informationsdarstellung eines Projektes erstellt werden, sodass alle Beteiligten ein verständlicheres und holistischeres Bild erhalten ?*

— *Mittels welcher Ermittlungsform können konsensuale Anforderungen am besten erhoben werden ?*

—*Welche Techniken und Technologien eignen sich am vorteilhaftesten zur Realisation ?*

—*Wie können die erwarteten Ergebnisse am geeignetsten praxisgerecht überprüft werden ?*

1.5.1 Erwartete Ergebnisse

Um den in der Praxis beobachteten Defizit der heterogenen Informationsrepräsentation gepaart durch teilweise Mißinterpretation, der durchaus aus unterschiedlichen „Domänen“ stammenden Beteiligten, zu adressieren, soll ein Software-Prototyp entstehen. Dieses Werkzeug soll einerseits die Kommunikation zwischen den Projektbeteiligten (vertikale Hierarchie) vereinfachen und andererseits eine Reifikation des betreffenden Informationsextraktes in probater Weise darstellen. Diese Thesenimplementation soll schlussendlich in einer praktischen Erprobungsphase systematisch verifiziert werden.

Ein Erfolg wird mittels Annahme beziehungsweise Verwerfung der jeweiligen Hypothese ermittelt. Als Kriterium dienen, wie in Kapitel 12 & 13 ausführlich

diskutiert, verschiedenste Verfahren. Die wichtigste Erhebung stellt die quantitative Auswertung dar, teilweise werden auch schwer prägnant zu formulierende Fragestellungen mittels einer qualitativen Untersuchung vollzogen. Die aktive Beobachtung, im Sinne der Feldforschung, wird stets unternommen um auch unvorhersehbare Effekte erfassen zu können.

Wichtig hierbei ist, dass jene Hypothesen in ihrer Formulierung lediglich den Stellenwert einer *bedingten Annahme* haben, und deswegen auch nicht im klassischen Sinne einer deskriptiven statistischen Verteilung unterworfen werden, wobei auch die Stichprobengröße bei Weitem zu insignifikant ist. Da dieses Forschungsgebiet einen inhärent interdisziplinären Charakter besitzt können nun drei Komponenten unterschieden werden, worin folgendes postuliert wird.

1.5.1.a Soziotechnische Komponente

Dieser Aspekt beleuchtet die soziale Veränderung aufgrund einer kollaborativen Entwicklung eines gemeinsamen „Fachjargons“. Es wird erwartet, bedingt durch die iterative Annäherung der heterogenen Gruppen, dass sich ein stärkerer Kontext im Sinne der gegenseitigen Betrachtungsweise bildet. Diese Komponente wird hauptsächlich mit einer qualitativen wie auch quantitativen Analyse festgestellt.

1.5.1.b Projektmanagement Komponente

Insbesondere der klassischen Kommunikationshierarchie innerhalb des Projektes werden hier vor allem strukturelle Veränderungen im Sinne einer zeitlich wie auch qualitativ optimierten Vorgangsweise erwartet, da ein gewissermaßen gemeinsames Grundverständnis gegeben durch die Formalaspekte des angewandten Systems gegeben ist. Diese subtilen Vorgänge, die zumeist auch sehr persönlicher Art sein können, werden prinzipiell mittels einer qualitativen Erhebung versucht zu erfassen.

1.5.1.c Ökonomische Komponente

Diese Komponente sollte einen positiven Effekt im Bereich ökonomischer Einsparungen bzw. optimierter Ressourcenallokation vorwiegend bedingt durch zeitnahe und exaktere Entscheidungshilfen zeigen. Besonders durch die Ableitung diverser Metriken im Sinne hochaggregierter Kennzahlen basierend auf den Quelldaten ergibt sich ein supplementäres Bild der wirtschaftlichen Situation. Diese Vorgänge werden vorwiegend mit der teilnehmenden subjektiven Beobachtung aus dem Felde bewertet, da hier der projektierte Zeitrahmen unverhältnismäßig zu kurz wäre.

1.5.2 Der wissenschaftliche Beitrag

Aufgrund der zuvor gewählten Forschungsmethodologie sollte diese Arbeit basierend auf ihren Ergebnissen einen regen Diskurs in den folgenden wissenschaftlichen Bereichen weiter vertiefen.

Das konsequente Erforschen der in der Praxis persistierenden Paradigmen, und das Ableiten gewisser Möglichkeiten um klassische Problemstellungen, wie labile Informationskanäle, inadäquate Informationsdarstellungen und ungewollte Informationsasymmetrie, zu mindern. Im Speziellen wird hier der Bereich der Organisationslehre und deren Kommunikationswege verstanden.

Einen kollaborativen Ansatz finden der den „common sense“ der Problemlösung darstellen soll. Die Managementlehre sollte hier angeregt werden, die teilweise hybride Behandlung übergreifender Bereiche kollaborativer zu betrachten.

Weiters wird durch einen benutzerzentrierten Ansatz im Sinne eines „Proof of Concepts“ das Erproben der These partizipativ anhand einer Referenzimplementation in der Praxis durchgeführt. Durch die Erfahrung der Verifikationsergebnisse in inkrementeller Weise das Konzept verfeinern beziehungsweise auch zu verwerfen wenn notwendig, ist die primäre Aufgabe des Softwaredesigns um in adäquater Weise Probleme wie Performance und Software-Ergonomie zu adressieren.

1.6 Gliederung

Die nachfolgende Dissertation ist hauptsächlich durch die zuvor aufgezeigten Intentionen motiviert. Besonders inspirierend waren vor allem die Forschungstätigkeiten im Bereich „Infrastructure for Managing Semantic Documents“ sowie des Projektes Tesseract welche prinzipiell geeignete Strukturen und Forschungsmethodologien vorgaben. Diese werden speziell im nächsten Kapitel behandelt.

Das Werk gliedert sich prinzipiell in 5 Teile.

— Teil I befasst sich vorwiegend mit dem Hintergrund der Arbeit und ähnlichen Forschungsstudien. Er streift zuerst kurz die geschichtliche Entwicklung von Versionskontrollsystemen und führt danach die wichtigsten Nomenklaturen ein und endet mit dem Rezipieren verwandter Publikationen.

— Teil II erschließt die These samt ihren Theoriebezügen die sich hauptsächlich interdisziplinär auf die Projektmanagementlehre und die Kommunikationswissenschaft stützt, wobei hierbei lediglich die essentiellen, der Arbeit dienlichen, Konzepte vorgestellt werden, da andernfalls der gesamte Theoriekomplex an sich den gefassten Rahmen der Arbeit sprengen würden. Der Verlauf dieser Thesenvorstellung schließt mit einer ontologischen Betrachtung der Entitäten des Modells und ergründet ferner die Motivation zur RDF Beschreibungssprache. Basierend darauf, wird zum weiteren Verständnis, der Begriff des RBMSS[1] eingeführt.

— Teil III diskutiert anschaulich die Referenz-Implementation, die als „Proof of Concept“ verstanden werden kann, und ergründet ferner detailliert die Implementierungsentscheidungen von RBMST[2], eine Instanz von RBMSS.

[1] RBMSS: Repository Management Support System
[2] RBMST: Repository Management Support Tool

— Teil IV knüpft mit dem Hypothesentest, der im Zug der Erprobungsphase exerziert wird, an. Nach dieser Vorstellung werden die Methoden der Untersuchung in quantitativer und qualitativer Hinsicht aufgezeigt. Die daraus entstandenen Resultate werden vorerst in Rohform dargelegt und anschließend ausführlich diskutiert. Zur teilweisen Untermauerung mancher Aussagen, werden fallweise angestellte Beobachtungen aus der Feldforschung herangezogen.

— Teil V fasst die gewonnen Einblicke in einem größeren Kontext zusammen und schließt ferner mit einem Ausblick über jene Punkte der These und ihrer Implementierung, die sich sinnvollerweise als weiter verfolg- beziehungsweise ausbaubar etabliert haben und auch jene die sich seitens der Industrie als zusätzlich wünschenswert ergeben haben.

Teil I
Der Hintergrund und theoretische Aspekte

2 Kapitel

Historischer Überblick und vergleichende Studien

„Alle Wissenschaftler versuchen,
an der Pyramide menschlichen Wissens weiter zu bauen.
Ich hoffe, dass ich einen kleinen Stein dazutun konnte.“
Stephen Hawking

Schwerpunkte in diesem Kapitel:

- Hintergrund
- Verwandte Arbeiten

2.1 Hintergrund

Das „Backbone“ der gesamten Forschungstätigkeit beruht auf dem Faktum, dass ein reicher Metadatenbestand jedem „Repository“ inhärent innewohnt. Bedingt durch die Verwaltung sekundärer Informationen wie Autor, Datum und Revision ergibt sich zwangsweise eine implizite Zusatzinformation zum Primärdatenbestand. In dieser These stehen Metadatenbestände von SCM[1]-Repositories im Zentrum der Betrachtung, da diese als Basis für die Kernaussagen der projektmanagementspezifischen Ableitungen dienen. Im Folgenden wird speziell auf die historisch bedingten Unterscheidungsmerkmale und deren Nomenklaturen eingegangen. Im Zuge dieser Abhandlung wird die Bezeichnung *Repository* oft synonym mit dem Plural *Repositories* und dem *Repositorium* gleichgesetzt und somit der ureigene Begriff auch etwas gedehnt.

Exkurs „Das Repository“

Unter diesem neudeutschen Wort, in altdeutscher und lateinischer Form Repositorium, versteht man in der Informatik digitale Datenspeicher die über einen bestimmten Indizierungsmechanismus die Artefakte, oder salopp formuliert lediglich die Daten, der Anfrage liefern. Meist spricht man in diesem Zusammenhang von Datensätzen, welche den logischen Verbund einer gruppierten Datensammlung entsprechen. Die Datenhaltung innerhalb des Repositoriums, in der Literatur auch synonym als *verwaltete Verzeichnisse* bezeichnet, kann auf vielfältigste Weise geschehen. Angefangen über einfache semantische Bezeichnungen auf Dateisystemebene, über SQL basierte relationale Datenbanken bis hin zu JSON[2]

[1]SCM: Software Configuration Management

notierten Dateien und schemalosen Dokument zentrierten Datenbanken wie *MongoDB*[1].

2.1.1 Software Configuration Management

Unter diesem Begriff wird die Konfiguration von Software im allgemeinen Sinn verstanden, dass weit mehr als eine bloße Versionskontrolle darstellt. Es ist damit das gesamte Potential der Supportsysteme gemeint, die der Softwareentwicklung dienlich sind, wie zum Beispiel „Issue Tracking Support", „Wiki Systeme" automatische Build Umgebungen sowie das VCS[2]-System an sich. Für eine exaktere Abgrenzung, siehe dazu [AB2010].

2.1.2 Source Control Management

Bis in die späten 1980'er Jahre hinein beschränkte sich die Versionskontrolle auf Verzeichnisse in der Dateisystemebene, obwohl der Kern der Softwareentwicklung neben dem Quellcode an und für sich auch Konfigurationsdateien für Systemprogramme wie Compiler, Linker und Buildsysteme benötigt, die ebenfalls verwaltet werden müssen, war und ist dies hier nicht Gegenstand der Betrachtung. Bedingt durch zunehmende Komplexitätsanforderungen wie weitreichende Softwareabhängigkeiten und die generelle Projektgröße wurde die Notwendigkeit einer integeren Versionskontrolle vakant.

Aufgrund der mittlerweile zahlreichen Interdependenzen[3] im Sinne der Projektaufgabenverteilung, bieten moderne VCS oft zahlreiche Schnittstellen zu den in der horizontalen Hierarchie gleichgestellten Supportsystemen, seien sie extern, oder auch intern über „builtin" AddOns angebunden. Vergleiche dazu „Veracity" und „Bazaar", zwei Vertreter dieser Gattung.[4] Es wird noch daraufhin hingewiesen, dass der Begriff des Software Configuration Management, kurz SCM und VCS in der Literatur oftmals synonym verwendet wird.

2.1.3 VCS-Repositories

2.1.3.a Zentrale Architekturen

Diese Form der Architektur ist der Stereotype der Versionskontrollsysteme und stellt auch die erste Evolution nach den „*Verzeichnissystem basierten Ansätzen*" dar. Sie ist bis heute noch sehr gängig und bedient sich zwei orthogonaler Konzepte.

---

[2] JavaScript Object Notation
[1] Siehe dazu http://www.mongodb.com/de
[2] VCS: Version Control System
[3] Abhängigkeiten
[4] http://veracity-scm.com
http://bazaar.canonical.com

Das erste Konzept ist die isolierte Gestaltungsmöglichkeit des Autors im Sinne der Zugriffsmöglichkeiten auf das Repository. Es gibt hierbei zahlreiche Authentifikationsmechanismen die diesen Zugriff regeln.

Das zweite Konzept bezieht sich auf die Konfliktvermeidung von parallel bearbeitenden Ressourcen. Ein „*Locking Mechanismus*“ zumeist auf Filesystemebene fungiert als Sentinel[1] ausgecheckter Ressourcen.

2.1.3.a.1 Revison Control System

Ein archetypisches System das Anfang der 1980'er Jahre von Walter Tichy an der Purdue University mit dem Ziel das kommerzielle Pendant SCCS universitätsintern zu ersetzen und gleichzeitig vorhandene Systemschwächen zu beseitigen. Aufgrund der quelloffenen Lizenz entwickelte sich Angang der 1990'er Jahre ein massiver Entwicklungsschub in der „open source Community“ welche das System mit zunehmender Funktionalität erweiterten. Eine der wichtigsten Funktionalitäten war die Netzwerkfähigkeit [TD2004], woraus mit dem CVS[2]-System die nächste Evolutionsstufe der VCS entstand.

2.1.3.a.2 Sub Version

Diese Instanz ist einer der gängigsten Formen heutiger Zeit. Der Anfang der proprietären Entwicklung begann um das Jahr 2000 mit dem primären Ziel die Metadatenhaltung, die bis dato ausschließlich unter Berkeley DB realisiert wurde, zu entkoppeln. Es ist ein modern gestaltetes VCS welches bereits auch eine grafische Oberfläche und, bedingt durch die starke Verwendung, eine hohe Performance[3] in der Verarbeitungszeit besitzt. Es unterscheidet sich von seinen Vorgängern hinsichtlich seinen neuen Architekturansatzes, welcher nicht die einzelnen geänderten Dateien als Revision auffasst sondern vielmehr das gesamte Projekt, sprich der logische Verbund aller Artefakte. Im Jahr 2009 wurde Subversion an die *open source Apache Foundation* übertragen.

2.1.3.a.3 Perforce

Ein kommerzieller Vertreter, ursprünglich 1995 unter der gleichnamigen Firma mit dem neuartigen Konfliktmodell des „copy-modify-merge“[4] entstanden, welches das Ziel hat gleichzeitiges Ändern von Ressourcen zuzulassen und beim Einchecken vorgang die vorhandenen Differenzen im Optimalfall automatisch zu *mergen*. Diese Vorgangsart versteht sich als logische Erweiterung zum pessimistischen Ansatz des klassischen File Locking Mechanismus und findet auch in anderen Versionskontrollsystemen häufig Verwendung. Abhängig vom Entwicklungsworkflow kann bei

[1] Wächter
[2] CVS: Concurrent Versions System
[3] Performance wird hier für einen allgemeinen Leistungsaspekt verwendet
[4] http://de.wikipedia.org/wiki/Versionsverwaltung#Copy_Modify_Merge

diesem VCS jederzeit jeweils auf die andere Vorgangsart umgestellt werden.

2.1.3.b Dezentrale Architekturen

Sie ist die konsequente Weiterentwicklung des zentralen Ansatzes. Man identifizierte die Singularität des Repository-Servers als Schwachstelle. Einerseits stellt eine zeitweise Überlastung, andererseits die Verfügbarkeit generell den „Archillesferse“ derartiger Systeme dar.

Es gibt natürlich auch Techniken wie das Load-Balancing welches potenziellen Überlastungen weitgehend vorbeugt, als Faktum bleibt jedoch das Manko der singulären Verfügbarkeit, welches sich nur mit erheblichem Aufwand, wie zum Beispiel einer Serverredundanz, beheben lässt. Somit geht dieses Konzept weiter und verfolgt den Ansatz verteilter Repositories. Durch diesen Ansatz ergeben sich gleich mehrere Vorteile.

Das Hauptziel der erhöhten Verfügbarkeit ist nun gewährleistet, da nun jeder Entwicklungsplatz ein potenzielles Repository darstellt. Ein weiterer Vorteil ergibt sich durch die neu entstandene Flexibilität welche somit mehrere Arbeitsweisen, sprich Workflows, innerhalb der Entwicklung zulässt. Durch die dezentrale Verteilung kann auch eine erhöhte Sicherheit an Redundanz und Konsistenz der Datenbestände impliziert werden.

2.1.3.b.1 Git

Ein anfangs von Linus Torvalds 2005 entwickeltes VCS mit dem Fokus den Workflow der Linux-Entwicklung zu integrieren. Aufgrund der starken Linux Community erfuhr das System eine schnelle Verbreitung und Anwendung. Im Laufe der Zeit wurde es um viele Funktionen angereichert. Als wesentlichstes Hauptaugenmerk gilt die erhöhte Effizienz und kryptographische Sicherheit der Versionsgeschichte.

2.1.3.b.2 Mercurial

Ein sehr beliebtes VCS, konzipiert von Matt Mackall zeitgleich mit GIT, um als Alternative zur Linux Kernel Verwaltung zu dienen. Es besitzt wie auch sein Konkurrent ähnliche Funktionalitäten, und bietet ebenfalls Kontextintegration sowie grafische Bedienoberflächen an. Der Interaktionsmechanismus ist bei beiden ähnlich wobei die Flexibilität von Mercurial subtiler wirkt. Abbildung 2.1 veranschaulicht die vielfältigen Interaktionsmechanismen des Versionskontrollsystems Mercurial. Es lässt sich hierbei die 3 stufige Konstellation zwischen einem fernen Repository, oft auch als Master-Repository bezeichnet, und der lokalen Kopie samt aktivem Arbeitsverzeichnis erkennen. Ein Unterschied zu den zentralen Architekturen wo es eine dediziert eigenständige lokale Repository-Instanz so nicht gibt, sondern das synchronisieren stets mit dem *fernen Server* erfolgen muss. Die nebenstehenden klein abgebildeten Operationen sind nicht von weiterem Interesse.

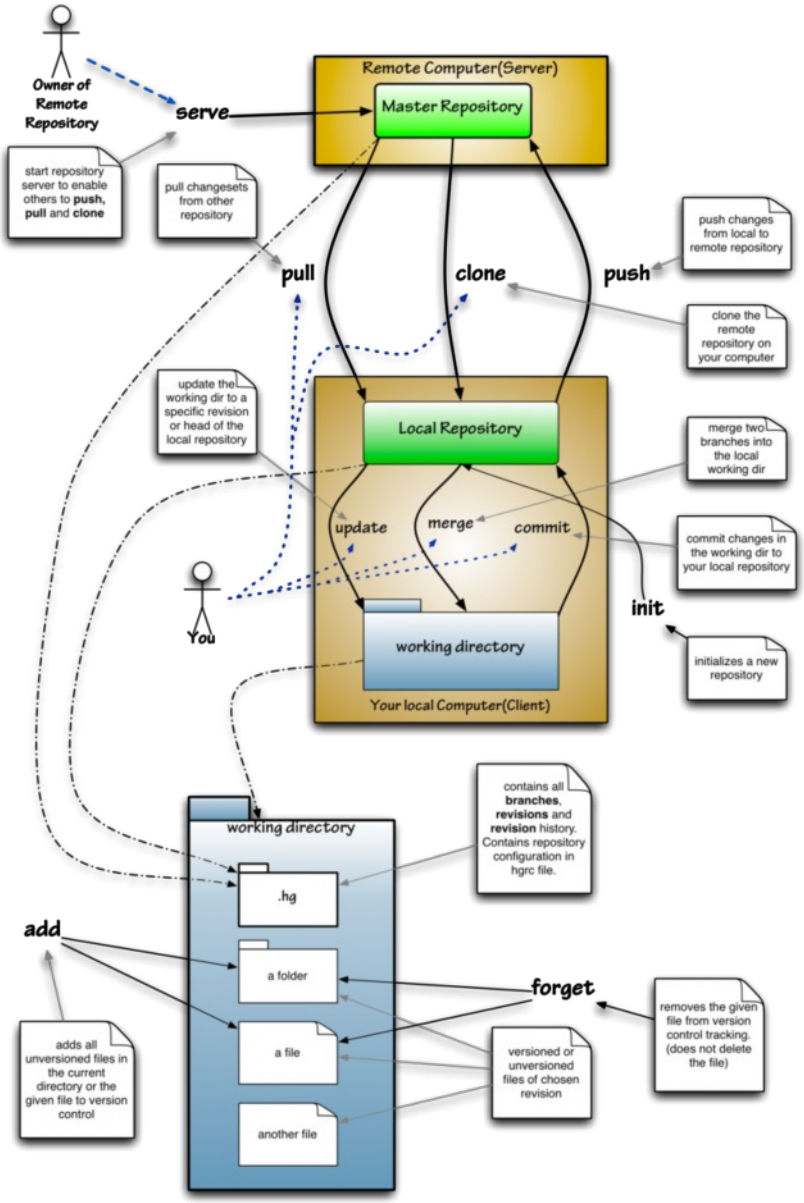


Abbildung 2.1 – Mercurial Interaktionsmechanismus

2.1.3.b.3 Darcs

Eine interessante Variante, da es einen neuartigen Weg im Sinne des „Auflösen von Merge-Konflikten“ bestreitet. Die sogenannte *Patch-Theorie* basierend auf Jason Dagits Arbeiten an der Tuft University[1] löst zuverlässig alle potenziellen Konflikte und garantiert eine autonome inkrementelle Patcherstellung zwischen den verschiedenen „Changesets“. Durch die Fokussierung auf Patchebene ergibt sich auch eine einfachere homogene Benutzerschnittstelle.

Es gibt noch eine Vielzahl interessanter Systeme, wie zum Beispiel „Fossil“ als monolithisches System in Bezug auf Kompaktheit und Vollständigkeit der Schnittstellen. Für eine vollständigere VCS-Taxonomie dazu, siehe [BS2005].

[1] http://www.cs.tufts.edu/comp/150GIT/archive/jason-dagit/tmr-darcs.pdf

2.2 *Verwandte Arbeiten*

Um Analogien in diesem thematischen Kontext zu finden, muss der Themenbereich holistisch erfasst werden. Durch diese Betrachtungsweise ergeben sich nun verschiedene Ansätze die auch in korrespondierenden wissenschaftlichen Arbeiten, meist als eigenständiges Projekt geführt, fußen. In den folgenden vier identifizierten Themenbereichen werden exemplarische jene Arbeiten rezipiert, die der Thesenbildung am dienlichsten waren.

2.2.1 Software Repository Mining

Besonders relevant für die gegenständliche Dissertationsarbeit ist hierbei einerseits der Ansatz der I*nformationsanreicherung durch Extraktion möglicher Metdadaten,* entweder über dem Dateisystem beziehungsweise auch über eine konsolidierte Erfassung zusätzlicher Medien wie E-Mail oder weiterer Supportsysteme.

2.2.1.a LibreSoft Research Group

Eine Forschungsgruppe der Universität Madrid mit dem Ziel „offener Kollaboration“ im Kontext der Software-Entwicklung und virtueller Gemeinschaften zu erforschen. Eine interessante Implementation namens „MetricsGrimoire“, damals noch unter den Titel „LibreSoft Tools“ bekannt, widmet sich folgendem Konzept. Den Kern der Suite bildet das Tool „CVSAnalY“[1].

„In CVSAnalY any interaction -also called commit- a commiter does with the central versioning system repository is logged with following data associated: commiter name, date, file, revision number, lines added, lines removed and an explanatory comment introduced by the commiter. There is some file-specific information that can also be extracted as for instance if the file has been removed. On the other hand, the human-inserted comment can also be parsed in order to see if the commit corresponds to an external contribution or even to an automated script. Basically CVSAnalY consists of three steps: preprocessing, database insertion and postprocessing.“ [RG2004]

2.2.1.b Rapid Miner Project

Hierbei wird der Ansatz deskriptiver statistischer Analyse in Kombination mit lernenden Algorithmen verfolgt. Ursprünglich im Jahr 2001 an der Universität Dortmund konzipiert, ist es mittlerweile eines der erfolgreichsten open source Projekte seiner Art. Durch einen kollaborativen Ansatz im Sinne einer „Cloud Lösung“ ermöglicht es Wissenschafter, Ingenieuren und Analytiker zugleich optimierte Unternehmensentscheidungen zu treffen.

Diese Implementation geht über den Horizont des reinen „Data Minings“ hinaus und interpretiert, nach einer Vorverarbeitungsphase zur Normalisierung der

[1] http://gsyc.es/~carlosgc/files/cvsanaly.pdf

Urdaten, statistisch gesicherte Werte. Es werden somit Informationen zur weiteren Analyse und pragmatischen Entscheidungsfindung bereitgestellt.

2.2.1.c Mining CVS repositories, the softchange experience

Das Interessante an dieser Vorgehensweise stellt prinzipiell die Extraktion von Daten aus einem holistischen Kontext dar. Hierbei werden sämtliche digitalen Spuren, ähnlich der digitalen Forensik, die als Fragmente eines Entwicklungsprozesses verbleiben, gesammelt und aufbereitet.

Es werden Einträge des CVS-Repository, etwaige Bugzilla Vermerke sowie korrespondierende Emails extrahiert und in einer weiteren relationalen Datenbank, dem „Softchange Repository“ zwischengespeichert. Danach wird ein separater Prozess angestoßen, um die sogenannten Softtrails in weiterer Folge zu visualisieren. Sei es über eine Vektorgrafik beziehungsweise Postscript Ausgabe die klassischerweise über die Browsertechnologie dargestellt wird. Abbildung 2.2 stellt die konsolidierte Metadatenextraktion verschiedenster Informationssysteme dar.

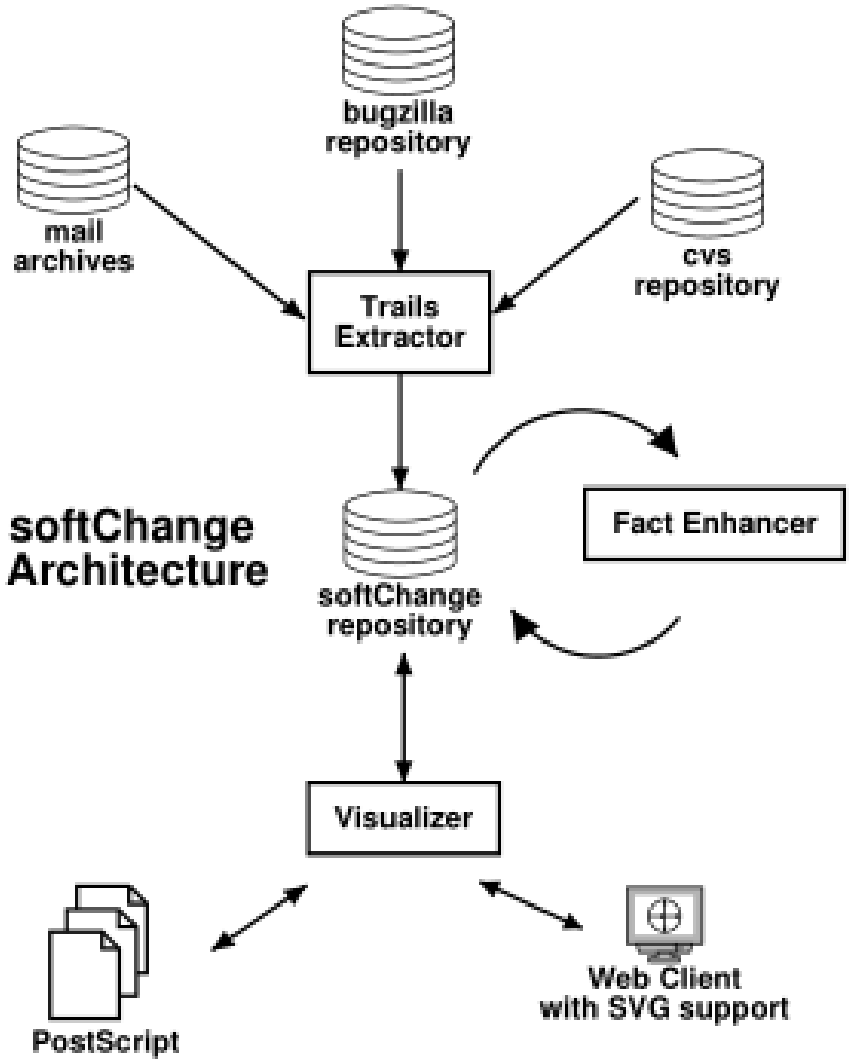


Abbildung 2.2 – Extraktion aus diversen Repositorien

2.2.2 Semantic Document Management

Bereichernd für die Dissertationsthese stellte sich hierbei vor allem die Idee der *logischen Speicherung softwarerelevanter Artefakte* sowie der Ansatz der individuellen Metabeschlagwortung. Ein weiterer Interessensfaktor bestand vor allem in der Realisation *domänenspezifischer* Werkzeuge zur gezielten Bewältigung des Vorhabens.

2.2.2.a Object Oriented Software Control Management

Diese Arbeit adressiert die grundsätzliche Problematik zwischen den diversen abstrakten Artefakten der Softwareentwicklung und der reinen dateibasierten Speicherung, in einem SCM. Um diese Dichotomie zwischen den vorwiegend logisch basierten Beziehungen zu schließen, wird das Versionierungskonzept hinsichtlich Objektorientierung erweitert. Damit benötigt das Arbeiten mit den logischen Objekten auch jeweils spezialisierte Tools wie separate Editoren und dergleichen.

Diese Anforderungen werden von der Beispielimplementation, dem Molhado Projekt [NT2004], adressiert, indem es eine Infrastruktur im Sinne einer „PlugIn Architektur“ anbietet wo das SCM sowie die dazu benötigten Werkzeuge integriert werden können. Das Ergebnis ist eine SCM zentrierte Entwicklungsumgebung, worin die logischen Objekte und Strukturen zuverlässig vorgehalten werden können.

2.2.2.b An Infrastructure for Managing Semantic Documents

Die Dokumentation von Software ist für alle Wissensträger der Softwareentwicklung gleichermaßen wichtig um in diesem Kontext erfolgreich kollaborieren zu können. Nichts desto trotz hat die Forschung gezeigt, dass gerade dieser Aspekt oft vernachlässigt wird. Ein Lösungsvorschlag bietet sich hierbei im Entwurf von Ontologien samt Anwendung semantischer Anmerkungen in Dokumenten an, um die nachträgliche Informationsbeschaffung zu erleichtern. Dieser Ansatz hat jedoch auch den Nachteil des zusätzlichen Zeitaufwandes der Annotation. Um nun auch dieses Problem zu bewerkstelligen, entwickelte eine Forschungsgruppe der Universität in Vitória eine Infrastruktur zum Anmerken und Verwalten von semantischen Dokumenten via Dokumentenvorlagen. Des Weiteren werden Versionsinformationen sowie Dokumentenänderungen an die Beteiligten notifiziert. [OA2010]

2.2.2.c Towards Document Repositories Based on Semantic Documents

In der Vergangenheit hat das Web die Nutzung von HTML-Dokumenten in praktischer Hinsicht etabliert, sie sind in pervasiver Form global verteilt und zugänglich. Um nun der Vielzahl an Dokumenten eine semantische Klassifikation zuzuordnen, darum bemüht sich eine Forschergruppe um Henrik Eriksson der Linköping Universität in Schweden. Den Ansatz den sie verfolgen ist klassischer Natur.

Sie entwickeln einen zuverlässigen Mechanismus der einerseits eine semantische Annotation druckbarer Dokumente, diese schließen binäre Formen wie PDF oder auch Microsoft Word Formate mit ein, gewährleistet, und auch die Abspeicherung und anschließende Auffindung jener „Semantic Documents“ zuverlässig realisiert. Es kommt dazu eine Ontologie, syntaktisch basierend auf OWL[1] zu tragen. Um diese Notation und Datenhaltung zu bewerkstelligen wird eine separate Toolchain dazu angeboten die dementsprechend auch die Suche semantischer Daten ermöglicht [EH2006]. Abbildung 2.3 zeigt die Interaktion der gedachten Toolchain mit den Benutzer und den dazu verwendeten Ontologien in einer Informationssuche.

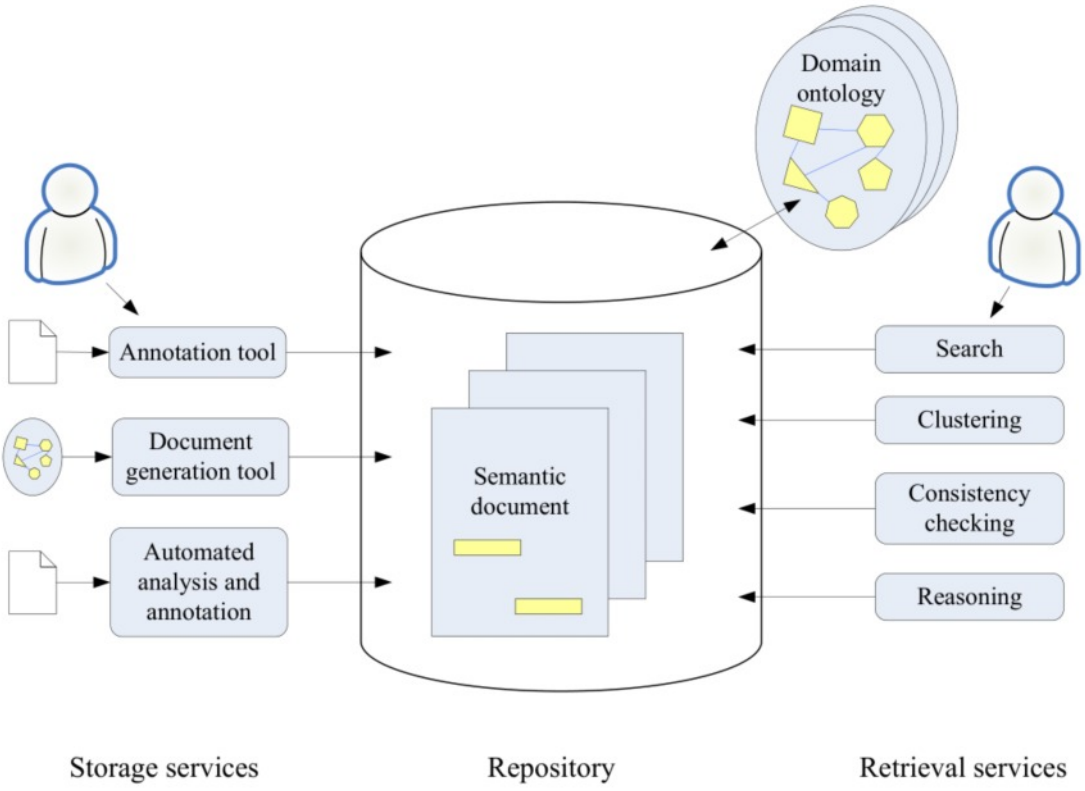


Abbildung 2.3 – Die Toolchain des Semantic Document Repository

2.2.3 Knowledge Extraction from Document Repositories

Hierbei kam vor allem die deutliche Akzeptanz von XML-Technologien zur Geltung, die als die de facto Basis zur Abbildung des „semantic Webs“ angesehen wird. Weiters war die Adressierung zur Problemminderung durch geeignete Algorithmen die eine Wissensdatenbank aufbauen sowie die *automatische Generierung firmeninternen Wissens* sehr lehrreich.

2.2.3.a Extracting knowledge from XML document repository

Es wird hierbei mittels einer „Proof of Concept“ Implementierung namens KROX der Versuch unternommen, in XML verfassten Dokumenten, zumeist Repositories auf die Ontologien OWL und RuleML basierter Daten, Informationen zu extrahieren die über den gedachten Einsatz wie Autorenschaft, Erscheinungsjahr und der

[1] eigentlich WOL: Web Ontology Language

gleichen hinausgehen um einen intra-organisatorischen Wert generieren, der letztendlich dem ergänzenden Aufbau des „semantic Web“ dienen soll.

2.2.3.b Collaboratively constructed knowledge repositories as a resource for domain independent concept extraction

Eine interessante Annäherung an intelligentes Management von unstrukturierten Informationen bietet diese österreichische Arbeitsgruppe. Es werden relevante Datenbestände in effizienter Weise gefiltert, da global agierende Unternehmen oft das Problem der Multilingualität in firmeninternen Korrespondenzen, zusätzlich auch aus unterschiedlichen Domänen stammend, haben. Mit dieser Möglichkeit kann oft zusätzlich generiertes unternehmensspezifisches Wissen in das tägliche Geschäftsleben einfließen. Dieser Prozess involviert die Bereiche künstliche Intelligenz, Text Mining und natürliche Sprachverarbeitung. Um solch eine domänenunabhängige Extraktion zu gewährleisten, ist für eine flexible Wissensgenerierung ein Aufbau eines „Knowledge Repositories“ unabdingbar um die firmeninternen Daten zu nutzen [KJ2012].

2.2.3.c Knowledge Extraction and Integration using Automatic and Visual Methods

Ein Framework-Modell das Techniken zur automatischen Ableitung neu gewonnen Wissens aus massiven Datenbeständen sowie Elemente der Visualisierungstechnik kombiniert. Es soll ein Prozess zwischen Wissensgenerierung aus semantischen Daten und menschlicher Expertise entstehen. Muster und Fakten die entweder durch Algorithmik oder visueller Analyse erkannt werden, sollen anschließend durch den Experten validiert werden. Im Erfolgsfall sollen die gültigen Wissensabbildungen in ein verlinktes Daten Repositorium integriert werden welches, angereichert durch visuelle Schnittstellen explorativ, nutzbar wird. Es besitzt starke Ähnlichkeit mit den Ideen des „social Web“, wobei hier der Fokus in Richtung der Aufbereitung einer Wissensdatenbank, ähnlich wie bei Wikipedia, geht. [SV2011]

2.2.4 Social Network Analysis

Diese Forschungstätigkeiten gaben Einblick in die vielfältigen Möglichkeiten der *Visualisierung von Relationen und Wissensverkettungen*. Des Weiteren war auch die Ergänzung bereits bestehenden Wissens durch zusätzliche qualitative Analysen wie Archivsuchen und ethnographische Erörterungen sehr richtungsweisend.

2.2.4.a Tesseract Project[1]

Nicht zu verwechseln mit den vielen gleichnamigen Projekten, wird hier die Tatsache belegt, wie auch die open source Entwicklung vorlebt, dass die Entwicklung von Software einen starken Konnex sowohl in der Technologie sowie auch in den sozialen Netzen zu verschiedenen Kontakten benötigt. Da dieses Thema

[1] http://www.cs.cmu.edu/~antz/tesseract.html

anfangs in der Forschung wenig Beachtung fand, wurde dieses Projekt von der CMU Universität erstmalig ins Leben gerufen.

Soziale Netzwerk Analyse hat gezeigt, dass verwobene Informationen bezogen auf die Verhaltensmuster und Beziehungen der Individuen im Sinne technischer Artefakte des Bereichs der Software Entwicklung sehr nützlich sind. Ein gleichnamiger Browser visualisiert den Beziehungskomplex in Bezug auf technische Gemeinsamkeiten der Kommune. Die Browserdarstellung mit ihrer Distanzmetrik erinnert ein wenig an die Erdös-Zahl[1], wo die Distanz im Graphen die Koautorenschaft ausdrückt. Dieses Tool wurde zu Demonstrationszwecken an der GNOME Projektdatenbank angewandt, anschließend mit Interviews ergänzt um die Durchgängigkeit des Ansatzes zu verifizieren [SA2009]. Abbildung 2.4 veranschaulicht beispielhaft die vielfältigen visuellen Darstellungsmöglichkeiten des Tesseract-Browsers.

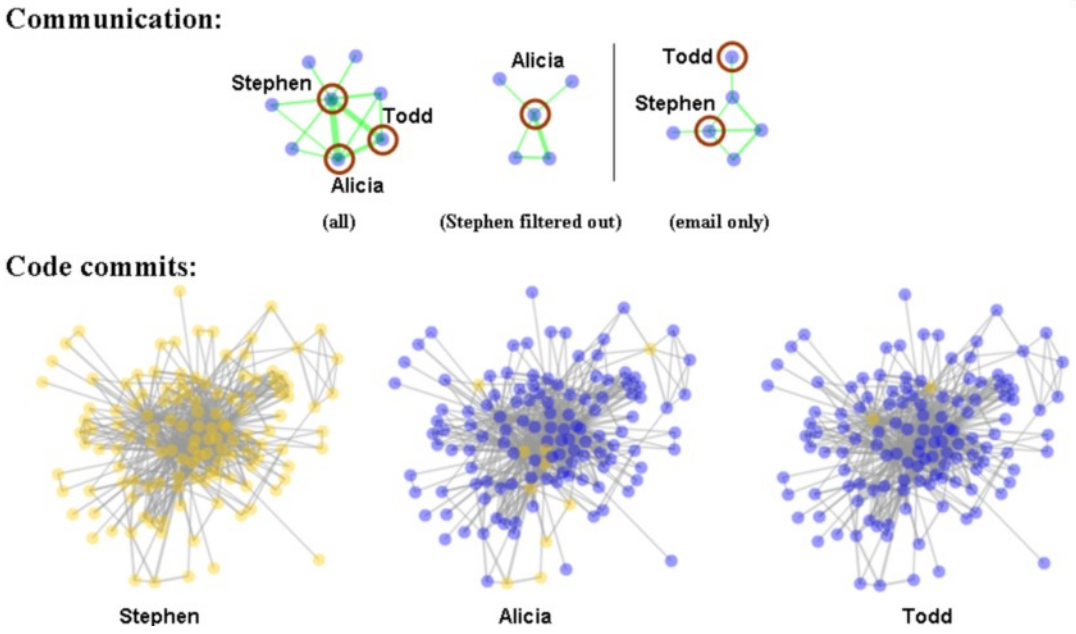


Abbildung 2.4 – Graph Visualierung über Tesseract Browser

2.2.4.b Mixed-method approaches to social network analysis

Diese Arbeit widmet sich der Frage nach einer qualitativen Bewertung der quantitativ ermittelten Daten sozialer Netze. In einem gemischten Methodenansatz werden beide Betrachtungsweisen miteinander kombiniert. Somit nähert man sich der Thematik mittels einer komplementären Ergänzung der zuvor gewonnen Daten. Es wird durch Literaturstudien belegt wie optimal sich qualitative und quantitative Methoden ergänzen, beziehungsweise mischen lassen. Durch geeignete Kombinationen wie Interviews, ethnographische und historische Archivsuchen ergibt sich somit, durch die differenzierte außenstehende Betrachtungsweise des Forschers, ein vollständigeres Gesamtbild des analysierten Netzes [EG2010].

[1] http://de.wikipedia.org/wiki/Erd%C5%91s-Zahl

Teil II
Die These und ihre Theorie-bezüge

3 Kapitel

Theoretische Betrachtungen und Klassifikationen des Projektmanagements

„There's nothing more practical than a good theory:
Integrating motivational interviewing and
self-determination theory"
Kurt Lewin

Schwerpunkte in diesem Kapitel :

- Das klassische IT-Projektmanagement
- Die Rollen und Mittelverteilung der Projektorganisation
- Supportsysteme der IT

3.1 Das klassische IT-Projektmanagement

Die Art und Weise wie Projekte geleitet werden ist vielfältig. Es gibt zwar klassische Vorgehensweisen die jedoch nie einheitlich anwendbar sind. Es haben sich natürlich im Laufe der Zeit gewisse „best practices" etabliert. Neuerdings unterscheidet man zwischen den klassischen Ansatz und einer agilen Form. Beides ist nicht konkurrenzlos und hat somit seine Vor- und Nachteile. Letztendlich verfolgen beide jedoch ein Ziel:

„Das Vorhaben unter den gegebenen Restriktionen erfolgreich zu bewerkstelligen, unter besonderer Berücksichtigung einer outputorientierten Ausrichtung."

Es bedeutet somit das Plansoll zu erreichen, und gegebenenfalls die dazu benötigten Inputs, sprich Ressourcen, zu minimieren. Eine gängige Veranschaulichung der wichtigsten Bedingungen eines Projektmanagements wird in Abbildung 3.1 dargestellt. Die Fläche des Rechtecks muss immer gleich groß bleiben, es können jedoch die Eckpunkte an ihren Diagonalen entlang gleiten, was wiederum eine Verschiebung des jeweiligen Bereichs wie Termin, Aufwand, Qualität und Leistungsumfang bedeutet. Diese Betrachtungsweise, auch unter dem Namen Teufelsquadrat[1] bekannt, ist jedoch primär eine ökonomische Ansicht und zeigt nicht die potenziellen Möglichkeiten der Realisierung auf. Speziell das Managen von IT-Projekten, im Besonderen von Software, stellt eine gesonderte Herausforderung dar [BH2008].

[1] Harry Sneed's Teufelsquadrat, es gibt auch dazu ein Teufelsdreieck

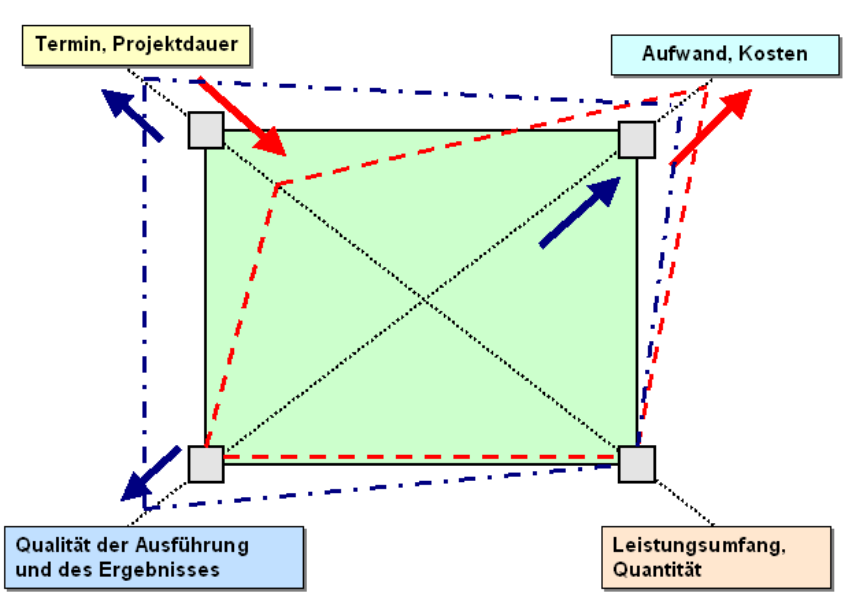


Abbildung 3.1 – Sneed's Teufelsquadrat

Es unterscheidet sich in der Regel dadurch, da Software

— immateriell ist

— ihre Eigenschaften sind schwer messbar/beurteilbar

— (externe) Fortschrittskontrolle ist schwierig

— erschwerte Kommunikation mit dem IT-Laien

— Verlauf ist "nicht stetig"

— geringe Änderungen haben oft gravierende Auswirkungen

— Umfeldabhängigkeiten sind wenig erforscht

— umgebungsspezifische Anpassung der Vorgehensweise notwendig

— wenig gesichertes Wissen über Vorgehensmodelle

— Zusammenhang zwischen Anforderungen und Kosten ist schwer vermittelbar

— Änderungen, die Anwender als „gering" einschätzen, haben oft gravierende architekturelle Konsequenzen

— gewisse „Unteilbarkeit" der Entwicklungsarbeit

— Zeit kann nicht beliebig kompensiert werden

Somit ist die Entwicklung von Software eine sehr interdisziplinäre Wissenschaft, da sie bedingt durch ihre konkreten Anforderungen mehrere Disziplinen beansprucht. Es ergeben sich Überlappungen in den Bereichen Projektmanagement,

Softwareerstellung und einen wesentlichen Teil stellt das Qualitätsmanagement, kurz QM, dar.

Der CHAOS Report, eine fortlaufende Studie von Erfolgen und Misserfolgen realisierter IT-Projekte, initiiert von „The Standish Group Inc.[1]“, zeigt erfreulicherweise eine kontinuierliche Verschiebung von „fehlgeschlagenen Projekten“ in Richtung „erfolgreicher bzw. herausfordernder Projekte“ an.

Dies stützt sich vorwiegend auf das Faktum der Anwendung besserer Methodiken im Projektmanagement, agilerer Vorgehensweisen, stärkerer Einbindung des Kunden sowie auch den Bereich der effizienteren Qualitätssicherung, die bis in 1990'er Jahre ein stets unterschätzter Faktor war. In Abbildung 3.2 werden die diversen Überschneidungen der Disziplinen kurz veranschaulicht, vergleiche dazu [HA2001].

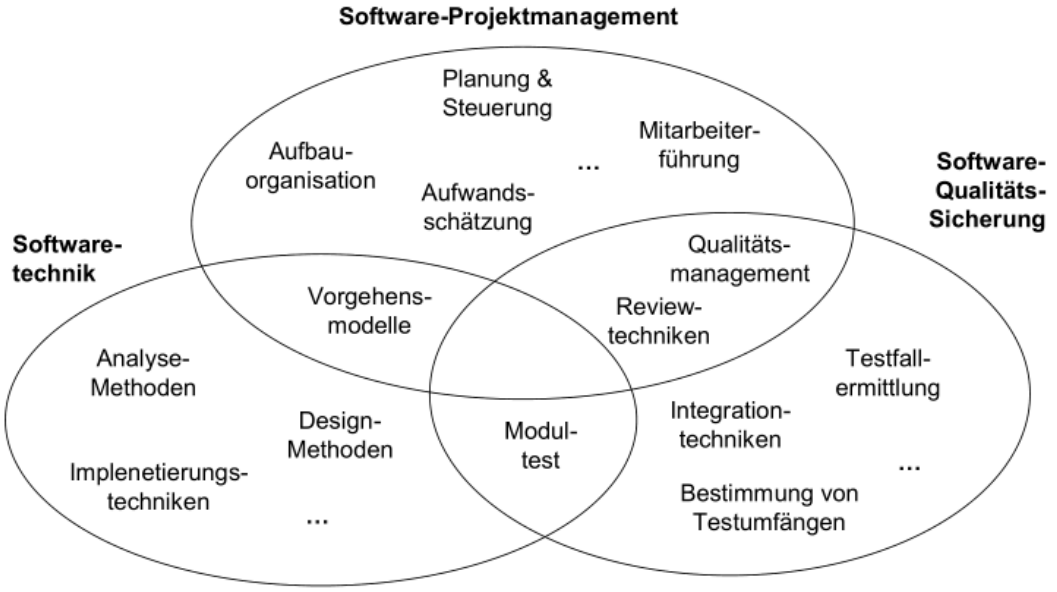


Abbildung 3.2 – Aufgaben des Software-Projektmanagements

Das Management konzentriert sich, prinzipiell auf eine Kosten- Nutzensicht bezogen auf die Technologieauswahl, welche wiederum den Produktlebenszyklus beeinflusst, da durch einen möglicherweise neuartigen innovativen Technologieeinsatz die Zeit der Entwicklungsphase sowie das Risiko der Umsetzbarkeit steigt, gepaart mit einer zunehmenden Volatilität der Kalkulationssicherheit.

Ein wichtiges Instrument zum Durchführen einer Kostenkalkulation während des Projektzyklus stellt die Lebenszykluskostenrechnung dar. Dieser Ansatz erfordert eine stetige Verknüpfung der faktischen Kosten mit den ursprünglichen Anforderungen. Diese Art der Lebenszykluskostenrechnung bezeichnet somit die Kalkulation aller Kosten, die ein Produkt während seiner gesamten Lebenszeit, sprich von der Entwicklung bis zur Entsorgung, verursacht. Basierend auf dieser Betrachtungsweise entstammen angewandte Methoden wie die „Design to Cost Methode“, ein in der Produktentwicklung verwendetes Verfahren, das die kostengünstigste Lösung für einzelne Komponenten bereits in dieser Produktlebenszyklus-

[1] Siehe http://www.standishgroup.com/

phase sucht und Folgeaufwendungen wie Vertriebs- oder Servicekosten berücksichtigt.

Bei der Total Cost of Ownership werden wiederum lediglich die Kosten aus Betriebssicht betrachtet. Das sind Aufwände, die während und nach der Nutzung eines Produktes anfallen.

„Die Lifecycle Costing Analysis oder Lebenszykluskostenanalyse enthält neben dem Ziel der Lebenszykluskostenberechnung auch die Beschreibung der Annahmen und Einschränkungen sowie die Nutzen und Risiken des Technologieprojekts.“[1]

Gängigere klassische Aufwandsschätzungsverfahren wären mitunter das COCOMO-Modell, Delphi-Verfahren sowie das Abschätzen durch Analogie der Komplexität. Eine interessante Herangehensweise an die Softwareabschätzung bietet die PERT-Schätzung, die noch durch deMarcos expliziter 3-Punkt Schätzung verfeinert wird.[DT2003]

Es gibt auch agile Methoden wie zum Beispiel das Abschätzen des bloßen Komplexitätsumfanges, und nicht des Zeitaufwandes, im Team empirisch ermittelt entweder durch sogenannte Scrum-Sprints, oder aber auch durch das agile COCOMO-II Verfahren. Auf die einzelnen Verfahren wird jedoch nicht dediziert eingegangen, der interessierte Leser sei an dieser Stelle an [ÖB2007] verwiesen. Zur Veranschaulichung, in Abbildung 3.3, die Phasen eines IT-Projektlebenszyklus nach [BM2013].

Abbildung 3.3 – Der IT-Projektlebenszyklus

Eines haben alle Aufwandsschätzungen jedoch gemein, den vorwiegend statischen Charakter des Modells sowie einer „a priori“ Bewertung eines geplanten Szenarios. Wichtiger jedoch erscheint eine betriebsstörungsfreie stetige Erfassung des *tatsächlichen* Aufwandes in beliebigem Detailgrad, um drohenden Missständen prophylaktisch entgegen wirken zu können.

Die Punkte auf die sich die These dieser Arbeit stützt, sind vor allem die klassischen Kommunikationsparadigmen und deren Probleme in der Ablauforganisation sowie der „unstetige“ Verlauf der Entwicklung und die daraus resultierende Beurteilbarkeits- beziehungsweise Entscheidungsprobleme von Software, da das

[1] Zitat von Dr.-Ing. Dipl.-Math. Michael Schabacker (2011)

Produkt, wie oben schon gezeigt, eine komplexe immaterielle Entität darstellt die sich nur schwer mittels formaler Methoden, in zeitlicher wie auch monetärer Hinsicht, abschätzen lässt.

3.2 Die Rollen und Mittelverteilung der Projektorganisation

So wie im vergangen Abschnitt die klassischen Betrachtungsweisen des Projektmanagements und deren Aufwandsschätzverfahren in Bezug auf die IT stattgefunden haben, werden wir hier die eher personelle Rollenverteilung samt dessen Kompetenzbereich abhandeln. Die Verantwortlichkeiten ergeben sich im Regelfall implizit durch die vorherrschende Projektorganisation, und diese prägt wiederum die Projektstruktur. Prinzipiell werden Organisationen zwecks Arbeitsteilung eingerichtet, damit im hierarchischen Kontext ein zielabhängiges Führen überhaupt realisierbar wird. So ergeben sich strategische, taktische und operative Ziele. Diese individuellen Planungsstufen stellten in abstrakter Form ein „Vorwegnehmen von Realität dar, wobei der Zufall durch Irrtum ersetzt wird“. Es ist auch wichtig in dieser Betrachtungsweise zwischen einer Aufbau- sowie Ablaufstruktur zu unterscheiden, da die Erstere den statischen Charakter der Organisation beschreibt und nicht den dynamischen Ablauf, der jedoch gerade im Bezug auf die Kommunikation von essentieller Bedeutung ist.

„Der Begriff Ablauforganisation bezeichnet in der Organisationstheorie die Ermittlung und Definition von Arbeitsprozessen unter Berücksichtigung von Raum, Zeit, Sachmitteln und Personen. Darunter verstehen wir die Gliederung der Tätigkeiten und Abläufe der Organisationseinheit in Prozesse zur Erfüllung ihrer Aufgaben.“ [BM2013]

3.2.1 Aufbauorganisationen

3.2.1.a Linienorganisation

Sie ist die klassische Organisationsform, in der die Aufteilung der Projekt- und Personalverantwortung auf funktionaler Ebene stattfindet. Diese Aufteilung ermöglicht somit ein einfaches Strukturieren des Projektes im Sinne von Subprojekten die diese organisatorischen Teilbereiche widerspiegeln. Jedoch werden überlappende Aufgabenbereiche, sogenannte Querschnittsaufgaben, gerne übersehen da gemäß dieser feingranularen Architektur kein Zuständigkeitsbereich abgegrenzt ist [SE2007]. Abgesehen davon, findet sich diese Struktur in mehr oder weniger ausgeprägter Form in jedem Unternehmen als *Primärorganisation* wieder.

3.2.1.b Die Einfluss-Projektorganisation

Gemäß [PG2008] auch als Stabsorganisation bezeichnet, unterscheidet sich von einer reinen Linienorganisation insofern, dass zusätzliche Stabsstellen, sprich Stellen die flankierend Querschnittstätigkeiten sowie auch Überwachungsaufgaben wahrnehmen. Hierbei muss jedoch beachtet werden, dass das Equilibrium zwischen

dem Stab und der Linie erhalten bleibt, da ansonsten der Projektablauf durch personelle Überbesetzung zusätzlich verlangsamt werden kann.

Grundsätzlich haben beide Formen der Linienorganisation maximal fachliche Weisungs- jedoch keine Entscheidungsbefugnis gegenüber dem Leiter der unterstellten Abteilung. Die Dauer dieser Organisationsform wird zumeist über die Projektdauer fortgeführt um den Effekt des Erfahrungswertes auf zukünftige Projekte zu übertragen.

3.2.1.c Matrix Projektorganisation

Eine ausgewogenen Mischung zwischen funktionaler Linienorganisation und produktorientierter Organisation mit dem Hintergrund die Weisungs- und Entscheidungsbefugnisse gleichermaßen symmetrisch aufzuteilen. Somit unterbindet sie einerseits den Stab-Linien Konflikt, schafft aber bedingt durch die zweite Dimension der Produktaufgliederung jedoch eine weitere Überlappung der Kompetenzbereiche, da es nun beide Betrachtungsweisen gibt, eine funktionale und eine produktbezogene Ebene. In Abbildung 3.4 wird die einzigartige matrixförmige Aufteilung nach unternehmens- und produktspezifischer Ausrichtung dargestellt.

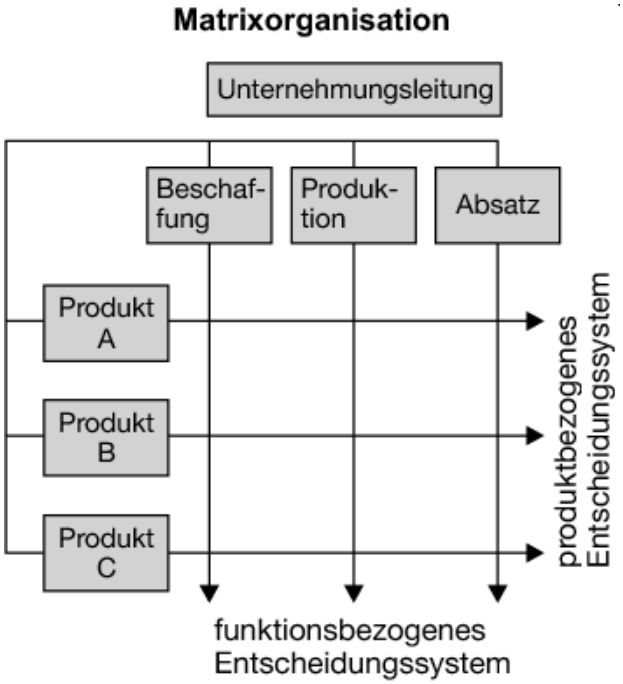


Abbildung 3.4 – funktionale und produktbezogene Aufteilung

3.2.1.d Pool Organisation

Sie stellt die dynamischste Form aller Organisationsformen dar. Der Hauptzweck ist es bei Firmen deren Kerntätigkeit das Projektgeschäft ist, keine starre Allokation der Ressourcen zuzuteilen. Die Projekte werden von temporären Projektmanagern geführt, die dafür benötigten Ressourcen dynamisch aus einem Pool benutzen können. Jeder Mitarbeiter stellt somit eine individuelle *Businessunit* dar, wo auch zugleich ein Nachteil dieser Struktur liegt, nämlich der Aspekt der Entlohnung da

hier einerseits nicht abteilungsweit gleich entlohnt wird, und andererseits die Intensität des Ressourceneinsatzes meist nicht ins Kalkül fließt.

3.2.1.e Reine Projektorganisation

Diese autonome Organisationsform, unterscheidet sich erheblich von den zuvor genannten Formen. Wie in der DIN 69901 definiert[1]:

"Ein Projekt ist ein Vorhaben, das im Wesentlichen durch Einmaligkeit der Bedingungen in ihrer Gesamtheit gekennzeichnet ist, wie z. B.: Zielvorgabe, zeitliche, finanzielle, personelle oder andere Bedingungen, Abgrenzungen gegenüber anderen Vorhaben und projektspezifische Organisation."

Gemäß dieser Forderung wird diese Struktur konzipiert. Die zeitliche Komponente ist begrenzt, da singulär, jedoch mit der vollen Verfügbarkeit und Zugriffsmöglichkeit aller projektspezifischen Ressourcen ausgestattet. Diese Form wird meistens bei sehr komplexen Projekten angewandt wobei der Projektmanager die Aufgaben im Interesse der Gesamtunternehmung führt. Er kann somit autonom über sämtliche Projektressourcen disponieren und ist lediglich einem Lenkungsausschuss unterstellt. Damit flacht die Hierarchie stark ab, und die Eigenverantwortung der Mitarbeiter wird verstärkt gefordert.

3.2.2 Ablauforganisationen

In der Organisationslehre wird unter diesem Begriff, das dynamische Abwickeln von Projekten verstanden. Es werden zu den in der Aufbauorganisation festgelegten Verantwortlichkeiten nun Abläufe und Aktivitäten zugeordnet, es findet somit implizit eine Arbeitsprozessbeschreibung statt. Hierbei wird zwischen einer Projektierung nach Phasen und Rollen beziehungsweise auch nach Produktstruktur unterschieden. In der Projektierung nach Phasen werden Expertenteams bezüglich des Fachbereichs eingesetzt. Im anderen Fall hingegen wird speziell ein auf das Produkt fokussiertes Team zusammengestellt. Der prozessorientierte Ablauf wird meistens innerhalb eines Teams wiederum segmentiert, wie am Beispiel des Gruppenleiters, der zwar die gleichen Kompetenzen besitzt wie ein reguläres Gruppenmitglied jedoch auch Schnittstellencharakter im Sinne des *Reportings* hat.

Eine besondere Rolle nimmt dabei der Projektleiter ein. Abhängig von der Aufbaustruktur ergeben sich Dispositionsfähigkeiten und Kompetenzen bereits im Vorfeld. Abgesehen davon hat er in der Regel grundsätzlich folgende Aufgabenstellungen zu bewältigen:[2]

— Erstellung des Projekt-, Termin- und Kostenplans

— Organisation und Koordination des Projektteams

— Durchführung und Fortschrittskontrolle

— Steuerung und Festlegung von Entscheidungen

[1] http://www.gerdneugebauer.de/PM2.pdf am 26.08.2014

[2] Vergleiche dazu Prof. Martin Wirsing 2008: Projektmanagement

— Personelle Betreuung der Projektmitarbeiter

— Mitwirkung bei der Projektzieldefinition

— Mitspracherecht bei der Bestimmung der Fachverantwortlichen

— projektbezogenes Informations-, Weisungs- und Entscheidungsrecht

Er stellt somit eine zentrale Figur im Projektablauf dar. Eine zentrale Kommunikationschnittstelle während der Projektdurchführung, vergleiche dazu [BM2013]. Abbildung 3.5 zeigt die vielfältigen Koordinationsbereiche eines Projektleiters und eines Qualitätsmanagers auf.

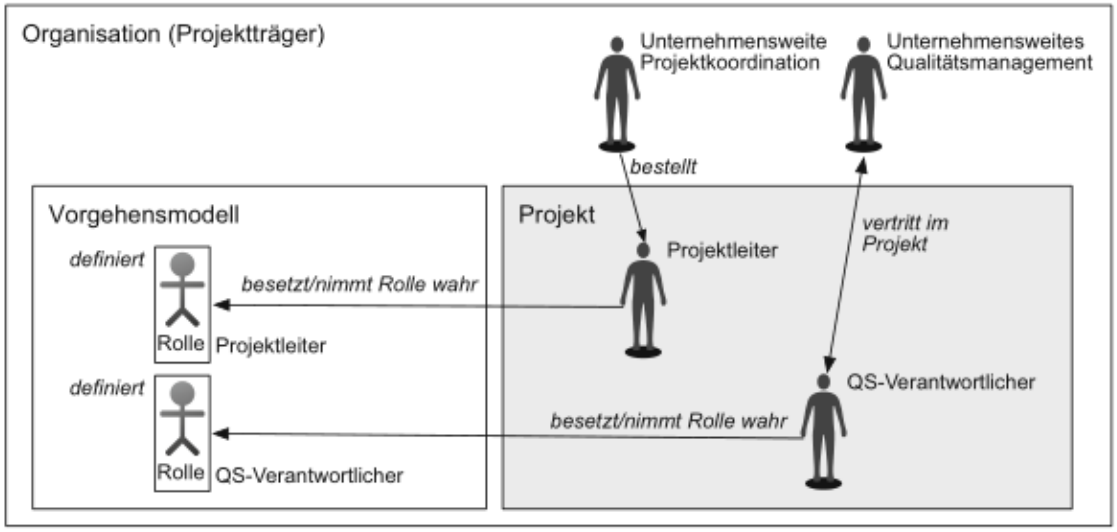


Abbildung 3.5 – Der Projektleiter als Schnittstellenfunktion

3.3 Support Systeme der IT

Um nun das Geschäftsmodell der Softwareentwicklung zu unterstützen bieten sich diverse Dienste im Verbund eines *Ecosystems*, gemäss folgender Definition an.[BJ2009]

„A Software Ecosystem consists of the set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions [Bosch, 2009]. Of course, a software ecosystem is also an ecosystem, specifically a commercial ecosystem, and hence the goods and services are the software solutions and software services that enable, provide support for or automate activities and transactions.“[1]

3.3.1 Versionskontrollsysteme

Wie schon im Kapitel 2.1.3 erläutert, spielt diese Form der verzeichnisbasierten

[1] http://www.software-ecosystems.com/Software_Ecosystems/Definition.html

Repositoriumsverwaltung eine zentrale Rolle in der Reihe der Supportsysteme. Es erlaubt einen sicheren Umgang mit Daten, da diese zumeist in Datenbanken gehalten, und mittels Integritätschecks gesichert sind. Die Unterstützung einer konsistenten Versionierungs- und Konfliktverwaltung stellt für den Entwickler an sich eine der wichtigsten Funktionalitäten dar.

Hierbei kann er im kollaborativen Sinne Modifikationen an der Software vornehmen ohne das Risiko eines Änderungsverlustes in Kauf nehmen zu müssen. Durch geeignete Konfliktmanagementwerkzeuge, sogenannte Diff-Tools, die Modifikationsüberlappungen visualisiert darstellen können, ist er in der Lage jeweils die korrekte Version eindeutig zu identifizieren. Durch das Paradigma der verteilten Versionssysteme ist er auch nun nicht zwingend an Ein- und Auscheckvorgängen anderer zentraler Repositories angehalten.

Moderne Versionskontrollsysteme bieten zudem auch vielfältige Bedien- und Interaktionsschnittstellen an. So zum Beispiel die Möglichkeit einer graphischen oder auch kommandozeilenbasierten Bedienung. Integrierte Netzwerkprotokolle wie ssh, http und https sind heutzutage Standard, welche somit auch eine einfache *Outsourcing*[1] Entwicklungen begünstigen. Ein wichtiger Nebenaspekt der Versionskontrollsysteme stellt der sogenannte Logmechanismus dar. Mittels diesem ist der Autor in der Lage, primär seine Änderungen zu dokumentieren, wie auch Verweise, kurze Quellenangaben und Verknüpfungen zu externen Systemen anzuführen. Um solche Zusatzinformationen zu propagieren, werden in Bezug auf „seperation of concerns“ wie unten angeführt andere Supportsysteme angeboten. Nicht selten jedoch bilden bereits moderne Versionskontrollsysteme Aspekte einer kombinierten Wiki/Issue-Tracking Lösung ab.[2]

3.3.2 Issue-Tracking Systeme

Bedingt durch die zunehmende Komplexität von Software sowie der Verwendung agiler Vorgehensweisen nimmt diese Systemklasse einen immer höher werdenden Stellenwert ein. Es kann hierbei unterschieden werden zwischen rein internen Systemen sowie auch Systemen mit einer extern benutzbaren Schnittstelle. Bei ersteren wird hauptsächlich die Plattform zur Dokumentation und Verfolgung von Softwarefehlern verwendet, wobei man auch zwischen bereits auf Kundenseite bekannten und unbekannten Fehlern unterscheidet, da sich hier meistens auch die Priorisierung in der Behandlung unterscheidet.

Jene Systeme mit externer Schnittstelle werden häufig dazu verwendet, dem Kunden eine Möglichkeit zur qualifizierten Adressierung eines festgestellten Fehlverhaltens zu bieten. Meist über eine grafisch geführte Eingabeschnittstelle hat er die Möglichkeit nicht nur Fehler, sondern auch projektspezifische Anfragen und Verbesserungsvorschläge zu deponieren.[3]

---

[1] Auslagerung einer Entwicklungstätigkeit an externe Unternehmen

[2] Vergleiche auch dazu http://www.fossil-scm.org

[3] Siehe dazu https://www.bestpractical.com/rt

3.3.3 Wiki Systeme

Diese Systemklasse unterscheidet sich von all den zuvor genannten Systemen insofern, da sie einen „social network“ Charakter aufweist. Hier ähnelt der Ablauf mehr einer kollaborativen Wissensbehandlung eines Themenkreises. Im Sinne des WORM-Paradigmas, „write once read many“ erstellt der Verfasser ein Hauptthema, welche Interessierte lesen und bei Bedarf um ihr kognitives Wissen ergänzen. Es stellt somit aus der Perspektive des Wissenswürfel ein kollektiv, explizit externes Wissen dar, welches in inkrementeller Weise wächst. [MG2010]

3.3.4 Hybrid Systeme

Unter dieser Bezeichnung wird auch manchmal eine Service Syndikation zusammengefasst. Es werden verschiedene Dienste miteinander verwoben und ergeben somit in Bezug auf einen Workflow, auch eine bessere Software-Ergonomie.

Das Hybrid-System kann als Verbundinstanz, die untergeordnete Subsysteme miteinander koordiniert, aufgefasst werden. Diese Art der Unterstützungssysteme verfolgt das primäre Ziel einer vereinheitlichten Bedienung über eine Schnittstelle. Einer der bekanntesten Vertreter seiner Art ist zum Beispiel die Open Source Software „Redmine“. Sie bietet selbst eine konsolidierte Web Schnittstelle zur Bedienung, vereint unter sich ein leichtgewichtiges „builtin“ Projektmanagement-system, „Wiki & Issuetracking System“ sowie diverse externe Schnittstellen zur SCM Systemintegration. Abbildung 3.6 stellt einen Aspekt des Projektmanagements in der Webapplikation Redmine beispielhaft dar.

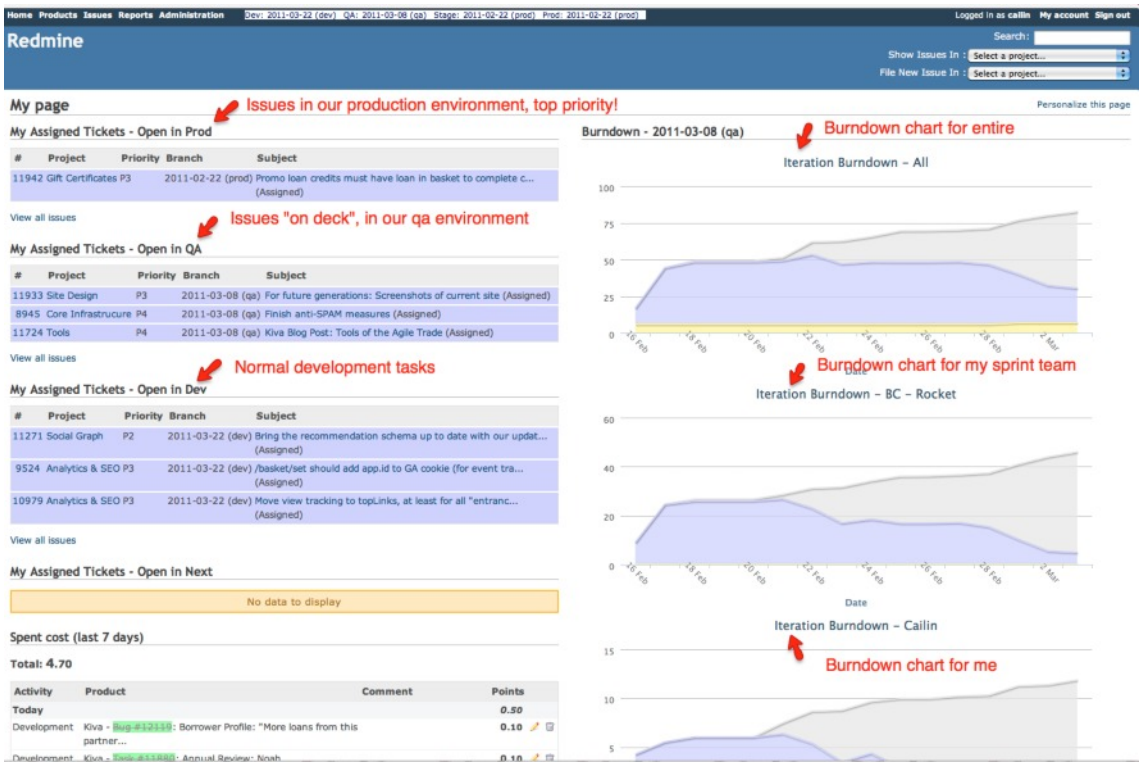


Abbildung 3.6 – ein vielseitiges Projektmanagementtool

4 Kapitel

Kommunikationsparadigmen und Einflussfaktoren im Projektablauf

„Man kann nicht nicht kommunizieren“
Paul Watzlawick

Schwerpunkte in diesem Kapitel :

- Strukturelle Probleme im Kommunikationsablauf
- Personenzentrierte Kommunikation
- Effekte und Gesetzmäßigkeiten im Projektablauf

4.1 Strukturelle Probleme im Kommunikationsablauf

Die Einbettung eines Projektes in eine Aufbauorganisation mit der anschließenden ablaufgetriebenen Projektdurchführung bedingt gefestigte Aufgabenverteilungen und Kommunikationsstrukturen. Abhängig von der Projektgröße steigt auch der Bedarf nach solider Kommunikation. Durch eine weiterführende Betrachtungsweise werden die Anforderungen auch durch die zunehmende Globalisierung und Dynamisierung des Wirtschaftsmarktes gestellt. [NK2012a]

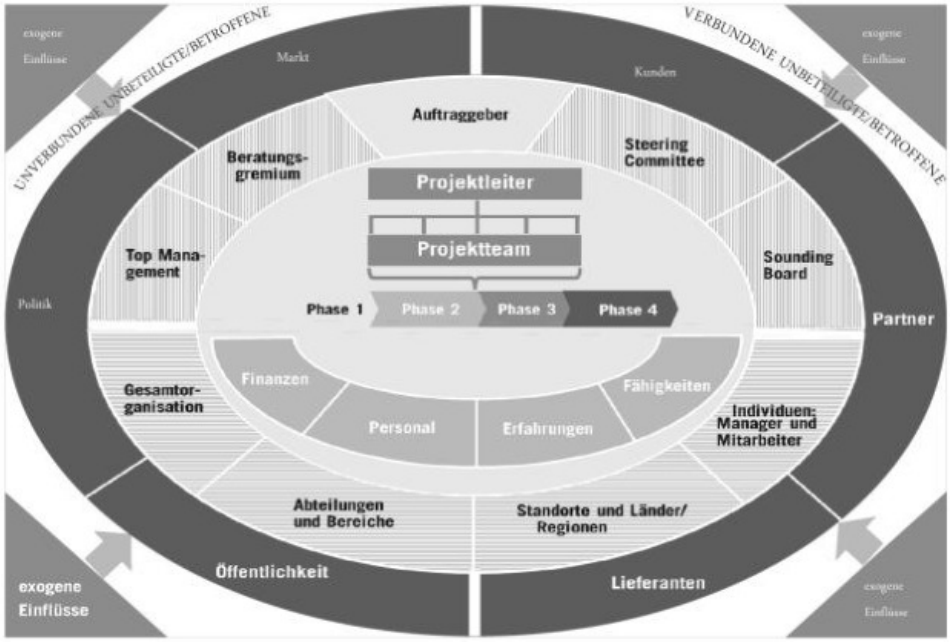


Abbildung 4.1 – Projektkommunikation und deren Steuerungen

Abbildung 4.1 veranschaulicht hierzu die vielfältigen Kommunikations- und Austauschbedürfnisse im Projektumfeld.

Einerseits erleichtert der Einsatz diverser elektronischer Kommunikations- und Informationseinrichtungen den Informationsfluss enorm, andererseits hingegen ist gerade die Informationsdichte in Bezug auf die transportierte Qualität fraglich. Über verschiedenste Medien gelangen Informationen an die Aktoren und überfluten das Individuum [LM2012a], geeignete Regeln und Filtermechanismen sind gefragter denn je. Überdies ist auch die Informationsquelle von Interesse, da Faktoren wie die Informationsaktualität und Symmetrie eine essentielle Notwendigkeit im rasanten Geschäftsleben darstellen. Generell wird die Qualität der Kommunikation im Gesprächsalltag zwischen den Akteuren stark von ihren Intentionen und Interpretationen beeinflusst [BJ2001].

Gemäß [NK2012b] können folgende Einflussfaktoren identifiziert werden:

Person des Senders	Seine Rolle und seine Macht, seine Situation und seine Absichten, nonverbale und symbolische Konnotationen, seine persönliche Glaubwürdigkeit, Integrität und Überzeugungskraft
Person des Empfängers	Die National- und Professionskultur des Empfängers, seine psychische Verfassung, seine Rolle und sein Status, seine Befindlichkeiten, der individuelle Erfahrungshintergrund
Inhalte	Ihre Ein- bzw. Mehrdeutigkeit, ihr Härtegrad, der Übertragungskanal und die gewählte Darstellung – bis hin zum Tod
Situation der Kommunikation	Ihr Kontext, das gesamte Umfeld, die Unternehmenskultur und Unternehmensgeschichte, Vorerfahrungen mit Sondersituationen, Öffentlichkeit der Kommunikation
Historie der Kommunikation	Vorausgegangene Interaktionen, gemachte Versprechen, gehaltene Zusagen, Vorgänger, Vorredner und ihre jeweilige Person
Individuelle Interpretation	Die individuelle Interpretation der Kommunikationsinhalte, des Kommunikationskanals, der Form, der Situation und des Kontextes, des Senders und seiner Absichten

Abbildung 4.2 – wichtige Einflussgrößen und deren Kommunikationswirkung

Wie in Abbildung 4.2 dargestellt, kann sich nun ein individuelles Projektverständnis einstellen. Ein hehres Ziel wäre jedoch sich um ein ganzheitliches Verständnis zu bemühen.

Einen besonderen Stellenwert nimmt die darstellungsbezogene Interpretation ein. Hier wird oft unbewusst, durch uneinheitliche Darstellungsformen desselben Kontextes, ein semantische Divergenz induziert. Das semiotische Dreieck[1] versinn-

[1] http://de.wikipedia.org/wiki/Semiotisches_Dreieck

bildlicht das Konzept, indem durch Symbolik eine differenzierte Betrachtungsweise im Sinne von direktem Objektbezug zu mittelbarer Begrifflichkeit beziehungsweise Gedankenabbild stattfindet, vergleiche dazu [SJ2009].

4.2 Personenzentrierte Kommunikation – ein Ansatz zur verbesserten Projektkollaboration

Ein paradigmatisches Diktum, basierend auf Carl Rogers Arbeiten, welches die Ansicht vertritt, sich in erster Linie mit sich selbst und dem „Gegenüber" auseinanderzusetzen. Es wird versucht seine Motivation und Denkweise zuerst zu verstehen, und dann zu agieren. Dazu bedarf es einer Kongruenz von, um nur einige der sogenannten Rogers Variablen zu nennen, Empathie, Aufrichtigkeit, Offenheit und Akzeptanz [MR2009].

Durch eine Online-Befragung zum Thema „zwischenmenschliche Kommunikation in der österreichischen ICT[1]" wurde ersichtlich, dass für 95% der Befragten, primär Manager und Absolventen der Wirtschaftsinformatik, erfolgreiche Kommunikation einen führenden Erfolgsfaktor in der Projektarbeit bedeutet. Eine weitere Untersuchung kam zu dem Ergebnis, dass eine qualifizierte Mehrheit von 39% den größten Lerneffekt durch Kollaboration im Team haben. Vorallem konstruktive Kommunikation in internationalen Teams erfordert mehrheitlich Ansätze der persönlichen Reflektion. Ein EU-weit ausgetragenes Projekt namens i*Com*[2] belegt eindringlich diesen Sachverhalt in einem breiten Kontext. [IT2014]

4.3 Effekte und Gesetzmässigkeiten im Projektablauf

Es sollte nun offensichtlich sein, dass Fehlallokationen der Ressourcen und suboptimale Kommunikation während des Projektverlaufes den Fortschritt und die Produktivität stark hemmen. Im Zentrum dieser Betrachtung stehen oft wiederkehrende Effekte die sich in den unterschiedlichsten Phasen manifestieren. Diese Erörterung dient der Veranschaulichung nach dem Kausalitätsprinzip – Ursache & Wirkung – im Projektmanagement. Es wird hier plastisch gezeigt, wie diese Effekte sich als eine logische Wirkungskette ergänzen, mit dem Hintergrund diese Ausprägungen in Bezug auf das Projektmanagement zu identifizieren und gegebenenfalls darauf angemessen zu agieren. Das im Zuge dieser Dissertationsarbeit vorgestellte Werkzeug kann als solch ein *Präventionsmittel* aufgefasst werden.

4.3.1 Adverse Selektion

Dieses Problem, hervorgerufen durch eine etablierte Informationsasymmetrie zwischen Anbieter und Nachfrager hinsichtlich Preis/Qualität, hat zur Folge, dass darauf basierte Entscheidungen in ökonomischer Hinsicht meist nicht pareto optimal getroffen werden. Dieses Prinzip wurde schon vor längerer Zeit festgestellt und durch das Akerlof'sche *„Lemons-Problem"* auf wissenschaftlicher Weise beschrieben [AG1971].

[1] ICT: Information and Communication Technology

[2] http://www.icomproject.eu/

Kurz formuliert: *„Kann ein Nachfrager die wahre Qualität eines Gutes oder einer Leistung erst nach Vertragsschluss tatsächlich bewerten, wird er seine Zahlungsbereitschaft generell an der durchschnittlichen zu erwartenden Qualität des Gutes oder der Leistung ausrichten. Für ein Angebot überdurchschnittlicher Qualität liegt der angebotene Preis beim Vorliegen dieser Informationsasymmetrie unter dem Grenzwertprodukt. Die jeweiligen Anbieter ziehen sich folglich vom Markt zurück, sofern sie Alternativen haben oder bieten eine mindere Qualität an.“*

In dieser Tradition kann auch eine entstandene Informationsasymmetrie im Bereich des Projektmanagement von IT-Projekten verstanden werden, wenn an und für sich kohärente Bereiche Informationen in inadäquater Weise zur Verfügung stellen. Dies kann bewusst aber auch unbewusst geschehen.

Im unbewusst herbeigeführten Fall, können temporäre Unverfügbarkeit von Informationen sowie nicht genügend verstandene Sachverhalte, bedingt durch eine volatile Datenbasis oder inadäquate Darstellung von Zusammenhängen, zu Fehlentscheidungen, wie grob falsch bemessene Projektzeiten oder schlechte Ressourcenauslastungen, verleiten. Dieses Zusammenwirken kann sich sogar negativ Verstärken indem die generelle Erwartungshaltung bezüglich der Informationsintegrität zunehmend sinkt und ad hoc Vermutungen anstelle gesicherter Informationen treten.

4.3.2 Bullwhip Effekt

Ein Phänomen, ursprünglich aus dem Bereich der Logistik stammend, beschreibt im Zusammenhang des IT-Projektmanagements den prinzipiellen Einfluss der Varianz der Kapazitätsauslastung im Verhältnis zur Projektdauer. Abbildung 4.3 veranschaulicht graphisch das Zusammenwirken der beiden Größen.

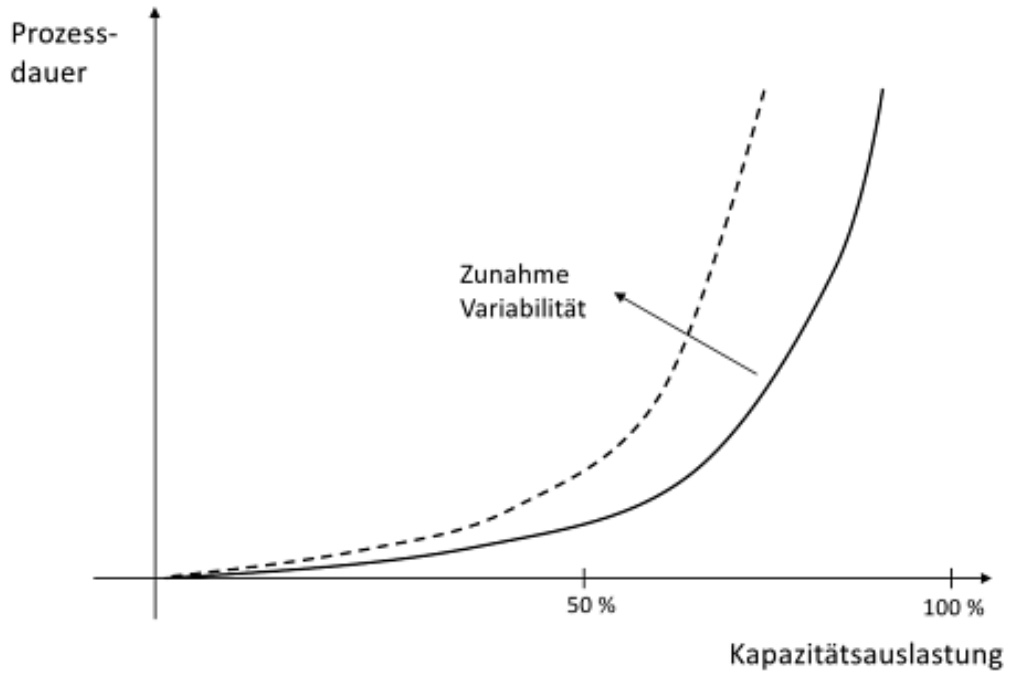


Abbildung 4.3 – Varianz zwischen Auslastung und Prozessdauer

Falls nun bestimmte Zeit Ressourcen überbeansprucht werden, entsteht in Bezug auf die projektierte Durchführungszeit ein zusätzlicher Zeitverlust. Dies kann entweder durch die nicht vollständige Beherrschung der eingesetzten Methoden beziehungsweise Technologien zurückgeführt werden, oder aber auch durch einen starken Verzug zwischen geplantem und tatsächlichem Projektstart [PP2014a].

4.3.3 Parkinson'sche Gesetz

Diese Gesetzmäßigkeit wurde bereits vor knapp 60 Jahren festgestellt und besitzt über das Projektmanagement weit hinausgehenden Anspruch auf Allgemeingültigkeit. Zuvor offenbarte der Bullwhip Effekt, dass eine hohe Varianz in der Projektplanung eine Störung im Gefüge der Produktivität der Kapazitäten darstellt. Im Konkreten kann eine Projektdauer nicht beliebig verkürzt werden, ohne in die „Impossible Zone", in der deutschen Literatur oft auch als Todeszone bezeichnet, zu geraten. Eine überdimensionierte Planzeit wird laut Parkinson aber stets ausgefüllt beziehungsweise konsumiert, egal wie viel tatsächliche Zeit ein Team für die Aufgabenbewältigung benötigt.

Das Team wird stets, die gesamte verfügbare Zeit beanspruchen, sprich es streckt das Zeitquantum insofern, dass es seine Leistung anders „taktet" wie zum Beispiel unbeauftragte Verschönerungen in der Software herbeiführt oder zusätzliche nicht vereinbarte Funktionalität integriert. Dies stellt jedoch nicht nur bezüglich des Projektes eine gewisse Fehlinvestition dar, sondern es gibt auch Fälle die Seitens des Kunden definitiv unerwünscht sind, man stelle sich nur die Boardsoftware einer Flugzeugtyps vor, die plötzlich unplanmäßig aussieht oder ungewohnt zu bedienen wäre.

4.3.4 Impossible Zone

Bei Parkinson wurde eine zeitliche Überdimensionierung vollständig aufgebraucht, aber eine zu starke Komprimierung der Projektzeit stößt auch an ihre Grenzen. Charles Simons und andere haben als erste eindringlich aufgezeigt, dass unabhängig der eingesetzten Aufwandschätzverfahren ein bestimmtes Ratio zwischen nominell *geschätzter Dauer : tatsächlich realistischer Dauer* nicht unterschritten werden kann, siehe dazu auch [GD2006].

Abbildung 4.4 stellt mit der Abszisse das zuvor gezeigte Zeitverhältnis, die Ordinate das proportionale Aufwandsverhältnis und die graue rechteckige Fläche hingegen den Unmöglichkeitsbereich dar. Der Schnittpunkt auf der Abszisse bei 1,0 stellt das optimale Ratio dar. Die geschätzte Dauer entspricht exakt der tatsächlichen Dauer.

Die Gerade größer 1,0 wiederspiegelt das Parkinson'sche Postulat der linearen Zeitausfüllung, andere Schätzmethoden zeichnen auch einen abfallenden Verlauf an. Offensichtlich kann der Wert auch kleiner 1,0 werden, welches einerseits eine bewusste Zeiteinsparung signalisiert, oder aber auch ein allzu oft praktiziertes Unterschätzen des tatsächlichen Projektaufwandes darstellt, der sich jedoch teilweise durch Einsatz zusätzlicher Ressourcen wieder kompensieren ließe.

Prinzipiell offenbart sich ein sprunghafter Anstieg im Verhältnis des Aufwandes, der sich jedoch negativer Skaleneffekte nicht ins Unendliche treiben lässt. Diese Beobachtung deckt sich mit den Ergebnissen diverser Softwareaufwandsschätzverfahren. In der Literatur wird mehrfach eine maximale Zeitkompression von ca. 25% angegeben.

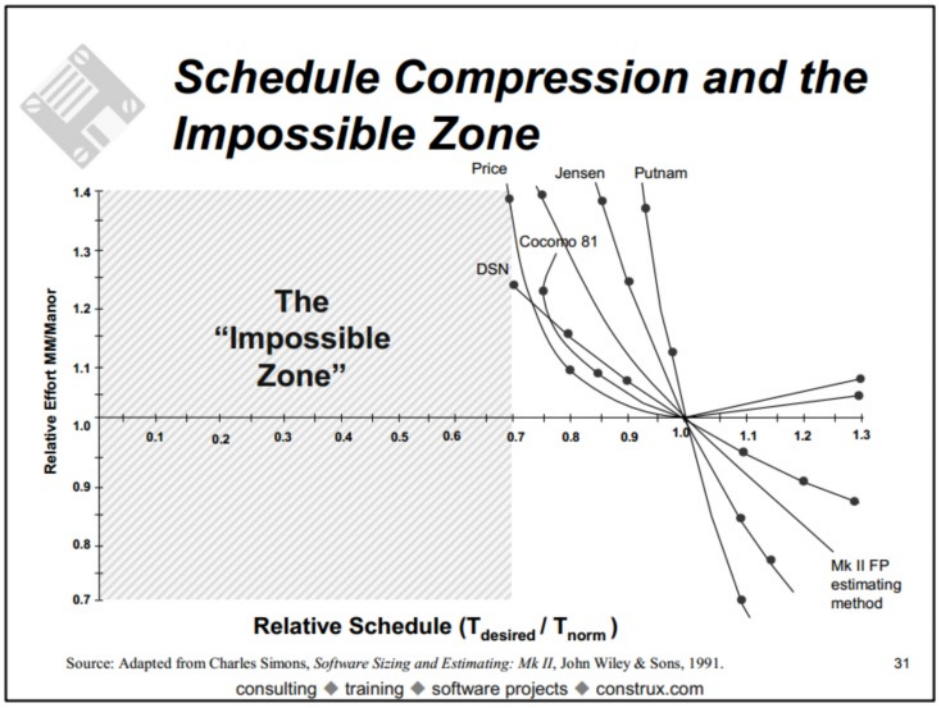


Abbildung 4.4 – die Todeszone der Aufwandsabschätzung

4.3.5 Brooks'sche Gesetz

Angeknüpft an den vorherigen Sachverhalt stellt sich nun die Frage ob und wann sich ein qualifizierter Zeitverlust durch einfache Ressourcenaufstockung der Entwicklungskapazität noch aufholen lässt.

Man kann festhalten, wenn sich das Projekt durch *n* Personen in der Zeit T=1 fertiggestellt werden kann, dann ergibt sich unter Voraussetzung gegebener Rahmenbedingungen, wie keine messbarer Zeitverlust durch Interkommunikation, bei n Personen ein dem entsprechend linearer Zeitaufwand von 1/n.

Findet zusätzlich Kommunikation statt, welche ja nicht nur eine Notwendigkeit darstellt sondern gewissermaßen eine Forderung ist, ergibt sich eine kombinatorische Problemstellung abhängig von der Anzahl der Beteiligten, wobei k einen Aufwandsfaktor pro Kommunikationskanal darstellt. Falls nun jeweils 2 Aktoren kommunizieren ergibt sich demzufolge ein Aufwand von

$$t \approx \frac{1}{n} + k\binom{n}{2} = \frac{1}{n} + k\frac{n(n-1)}{2}$$

Abbildung 4.5 – Formel des gesteigerten Kommunikationsoverhead

Es ergibt sich aus Abbildung 4.5 ein mathematischer Term, der einen Schnittpunkt von zwei Kurven, wie in Abbildung 4.6 darstellt, der das Optimum bezüglich des Humanressourceneinsatzes darstellt[1].

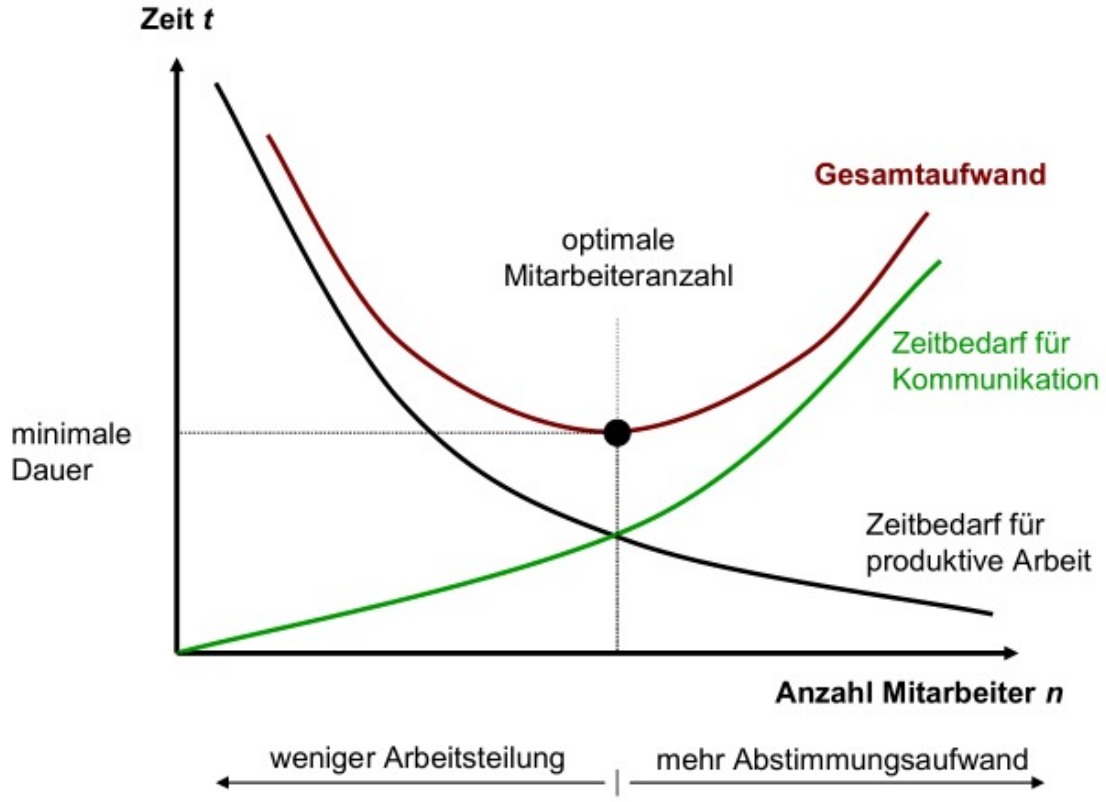


Abbildung 4.6 – Optimale Ermittlung der Mitarbeiteranzahl

Dies widerspiegelt in seiner Art abschließend Frederick Brooks's Gesetz:

„Der Einsatz zusätzlicher Arbeitskräfte bei bereits verzögerten Softwareprojekten verzögert sie nur noch weiter."

4.4 Theoriebezüge zur Thesenbildung

Die bereits in Kapitel 3 beginnenden Theorien- und Modellvorstellungen sollen hier nochmals in Bezug zur eigenen Thesenbildung harmonisiert werden. Prinzipiell basiert die These zur „Verbesserung der Informationsdarstellung im IT-Projektmanagement" darauf, dass bei der Verwaltung komplexer Projekte immer wieder verschiedene Kongruenzen in Einklang gebracht werden müssen, um so ein generelles *Projektequilibrium* zu erhalten. Die Schwierigkeiten die sich hierbei stellen sind jedoch Mannigfaltig.

In Kapitel 3 wurden im Zuge der prinzipiellen organisatorischen Projektteilung die Vor und Nachteile aufgezeigt. So stellen gerade in Bezug auf die These, abhängig

[1] Quelle: Dr. Frank Houdek, Management von Softwareprojekten, 2010

von der gewählten Projektstruktur, punktuelle Überlastungen einzelner Projektverantwortlicher starke Engpässe im generellen Projektablauf dar.

Das zu generierende Softwaretool soll dazu dienen, im Bereich der Informationsbeschaffung, derartige Engpass- und Unverfügbarkeitssituationen teilweise zu entspannen. Mittels der schnellen und einfachen Bedienung des Werkzeuges sollen einfache Informationszusammenhänge visualisiert werden, um so den Nutzer zeitlich zu entlasten. Abhängig von der gelebten Ablaufstruktur und der betriebenen Granularität der Informationsdarstellung können sich Verbesserungen im Bereich der effizienteren Ressourcenallokation, besseren Projektlastverteilung und dementsprechend auch psychosozialen Entspannung einstellen.

In Kapitel 4 werden vorwiegend dynamische Aspekte veranschaulicht, sie ähneln sinngemäß Faktoren, wie dem informationstheoretischen Rauschen eines Nachrichtenkanals, dadurch werden hier schwerpunktmäßig Kommunikations- und Interpretationsprobleme erörtert. Des Weiteren gibt es aber auch ressourcenrelevante Aspekte, die es während eines Projektablaufes zu kontrollieren gilt. Die dabei auftretenden Effekte und Gesetzmäßigkeiten sind sinngemäß Indikatoren die einer präventiven Alarmstufe gleich kommen.

Das in dieser Arbeit vorgestellte Werkzeug sollte somit in ressourcenbezogener und in zeitlicher Manier mit dem geplanten Projektverlauf a priori abgestimmt werden und dementsprechend eingesetzt werden, um so diversen Projektschieflagen entgegensteuern zu können. Durch den kollaborativen Entwurf der Informationsextraktion und deren anschliessende visuelle Aufbereitung kann eine Verbesserung der Kommunikation, der Interpretation wie auch eine erhöhte Informationssymmetrie erwartet werden.

5 Kapitel

Identifikationen und Forschungskonzeption

„Probleme kann man niemals mit derselben Denkweise lösen,
durch die sie entstanden sind.“
Albert Einstein

Schwerpunkte in diesem Kapitel :

- Projektabgrenzungen
- Problemidentifikation
- Das applikative Forschungskonzept

5.1 Projektabgrenzungen

In den vorangegangenen Kapiteln wurden die Grundlagen aufgebaut um die Notwendigkeit eines möglichen Lösungsansatzes, der Teilprobleme der unter Punkt 5.1 angeführten Aufzählung zu adressieren versucht, nachvollziehen zu können. Der daraus entstandene Lösungsansatz wird anschließend im Sinne der Methodenbildung als „applikatives Forschungskonzept“ dargelegt, dementsprechend eine konkrete Softwareimplementierung folgt. Der Terminus des IT-Projektmanagement wird hier in weiterer Folge zur Abhandlung abgegrenzt.

In Anlehnung an die Definition der DIN 69901[1] wo das

> *„Projektmanagement ist die Gesamtheit aller Führungsaufgaben, -techniken und -mittel für die Abwicklung eines Projekts.“*

definiert wurde, verstehen wir darunter auschliesslich ein Projektmanagement aus dem Bereich der IT[2] stammend, mit reiner Ausrichtung auf Softwareentwicklung. Die jeweiligen Aktoren werden hierbei im wesentlichen auf zwei Benutzergruppen beschränkt, das operative Management und die Implementatoren.

Des Weiteren wird ein Szenario vorausgesetzt, welches eine Arbeitsteilung mit interpersoneller Kommunikationskette verlangt. Somit werden Einzelprojekte, wo das Projektmanagement gleich dem Projektleiter und dem Entwickler ist, nicht berücksichtigt, da hier eine Problemstellung als rein *selbstreflektiv* zu betrachten wäre, und der hier vorgestellte Lösungsansatz nicht adäquat anzuwenden ist.

---

[1] DIN: Deutsches Institut für Normung
[2] IT: Informationstechnologie

5.2 Problemidentifikation

Allgemein erhebt die These, vor allem im Bereich des Softwareprojektmanagements, Anspruch auf vielfältigste Voraussetzungen wie

— Hohe IT-Kompetenz in fast allen funktionalen Bereichen

— Informationssymmetrie

— optimale Kommunikation

— konstante Belastbarkeit[1] der Ressourcen

— ex ante Abschätzungen des Softwareaufwandes

— hoher Grad an Ressourcen Substituierbarkeit

— *V*erfügbarkeiten diverser Unterstützungssysteme

— Ressourcenexklusivität im Projektkontext

um nur die wichtigsten zu nennen, wobei diese Aufzählung längst keinen Anspruch auf Vollständigkeit erhebt. In meiner beruflichen Laufbahn hingegen, habe ich erfahren, dass viele jener Komponenten zwar pauschal erwartet werden, jedoch in der Praxis weit weniger präsent sind.

Dies begründet sich wiederum aus den vielfältigsten Gegebenheiten, eine der hauptsächlichsten Treiber wären sicherlich allen voran Zeitmangel, Budgetrestriktion sowie den nicht zu unterschätzenden Faktor der „egoistischen Karriereplanung“ samt seinen individuellen Absichten.

5.2.1 Hohe IT-Kompetenz in fast allen funktionalen Bereichen

Eine gängige Fehlannahme ist jene, dass zwar viele involvierte Projektbeteiligte in ihren Domänen Experten sind, jedoch teilweise wenig bis zu wenig Querschnittswissen in den (vertikalen) Bereichen der Projektorganisation besitzen. Es wird dabei der Begriff eines gemeinsamen Projektverständnisses zu wenig „verinnerlicht“. Viele der Teammitglieder sind zu wenig bemüht einen breiteren Kontext zu sehen und verlassen sich implizit auf andere Instanzen.

Dies hat zur Folge, dass sich im Laufe des Projektfortschrittes eine sogenannte „semantische Kluft[2]“ bildet, die dann nur sehr schwer zu schließen ist, vergleiche dazu [MT2010]. Ein Faktum dem teilweise mit agilen Methoden, wie zum Beispiel den „brown bag sessions“, versucht wird entgegenzuwirken.

5.2.2 Informationssymmetrie

Dieser Faktor ist von entscheidender Bedeutung und wurde schon oft in diversen

[1] Hierbei wird vor allem auch der Stressfaktor subsumiert

[2] In der Literatur auch als „semantic gap“ bezeichnet

Abhandlungen ausführlich diskutiert [SH2010]. Das Problem, das sich hier stellt ist die theoretische Ansicht, dass jeder Projektbeteiligte bestimmte „projekthierarchiebezogene“ Rahmenbedingungen wie Termine, seien es Meilensteinmeetings oder Modulabgabefristen, hochpriore Change-Requests, dynamische Kundenzusagen in der laufenden Projektphase, neu entstandene Subprojekte und deren mehr, den gleichermaßen selben Informationsstand darüber hat.

Es sollte, wie vorhin angedeutet, zwischen den Symmetrien in horizontaler und vertikaler Hierarchieebene unterschieden werden. Auf horizontaler Ebene sollte der Informationsstand gleich sein, man könnte sagen eine Gruppenkohärenz bilden. Im vertikalen Bezug jedoch ist eine gewisse Asymmetrie von Nöten um bestimmte Unternehmensziele beziehungsweise Projektrestriktionen überhaupt verfolgen zu können, sprich die Informationskopplung ist hier weit loser zu sehen. Vorwiegend in der horizontalen Hierarchie, zeigt die Praxis hier jedoch oft ein anderes Bild.

Es sind zumeist Trivialitäten die diese Asymmetrien entstehen lassen. Oft sind sie einfach bedingt durch Abwesenheit, wie durch Krankheit, Urlaub oder auch längere Geschäftsreisen, die die individuelle Modellvorstellung des Projektes verrücken lässt. Im Trubel der Zeit wird diese grundsätzliche Holschuld der Informationsaktualisierung oft vernachlässigt, und man widmet sich einfach wieder dem Tagesgeschäft.

5.2.3 effizientere Kommunikationspfade

Mehr als nur spärlich, werden Informationen außerhalb regelmässiger Terminisierung ad hoc und in hohem Detailgrad gefordert. Es kommt öfters vor, dass eine plötzlich sehr spezifische Kundenanfrage an die Projektverantwortlichen gestellt wird, sei es weil der Kunde während der Projektphase seine eigene IT unvorhergesehen umstellen muss oder aber weil das strategische Management ungeplanterweise plötzlich großen Informationsbedarf hat, da die Eigentümer die Unternehmung veräußern wollen.

Diese Gegebenheiten bedingen allesamt eine effiziente Kommunikationsstruktur. Es sollte stets der kürzeste Pfad zur Informationsbeschaffung vorhanden, alle benötigten Informationsquellen verfügbar und die Information sollte auch vollständig sein. Diese Forderung ist nicht immer leistbar, da zum Beispiel konkrete Wissensträger einstweilig nicht verfügbar sind oder aber auch die Informationsquelle nur schwer zugänglich ist.

5.2.4 ex ante Abschätzungen des Softwareaufwandes

Es wurde schon im Kapitel 3.1 kurz vorgestellt, dass eine Vielzahl der verwendeten Abschätzmethoden einen *ex ante* Charakter aufweisen. Diese Betrachtung ist klarerweise notwendig, da sie ja zur Schätzung des Aufwandes *vor Projektbeginn* herangezogen werden. Problematisch hingegen ist, das statische Bild das sie vermitteln.

Nur selten werden in der Praxis formale Verfahren *während* der Projektphase angewandt um einen akkurateren Soll-Ist Vergleich führen zu können. Allzu oft werden Metriken *ex post* zu neuen Abschätzung via Analogie herangezogen, ohne die näheren Hintergründe der dynamischen Zusammensetzung der Metrik zu analysieren, wieder angewandt. Hier wäre eine bedarfgesteuerte detailliertere Korrelationsanalyse von Vorteil, bedingt jedoch auch eine fein granulare Auflösung der Aufwände auf Projektebene.

5.2.5 Hoher Grad an Ressourcen Substituierbarkeit

Ein vor allem in der Softwareentwicklung gängiger Irrtum stellt folgende naive Sichtweise des höheren Managements dar. Es wird angenommen, dass jede Ressource, somit auch Humanressourcen, einfach austauschbar sind. Das irrige dieser Annahme ist nicht die Verfügbarkeit, sondern vielmehr die Individualität der Arbeit des Vorgängers.

Die Softwareentwicklung besitzt trotz aller formalen Auflagen wie „Coding and Documentation Rules“ einen *höchstpersönlichen* Charakter. Es ist vor allem die Kenntnis programmatischer Idiome und das individuelle *Systemwissen* welches den jeweiligen algorithmischen Lösungsansatz vorgibt. Diese Summe an Kenntnissen ist unter firmeninternen Kollegschaften nicht gleich verteilt. Noch viel weniger Kenntnisgleichstand gilt unter externen Kapazitäten, wo ein weiterer Faktor der, der unterschätzten Einarbeitungszeit, noch hinzukommt.

5.2.6 konstante Belastbarkeit der Ressourcen

Die Idealvorstellung einer starren Leistung unabhängig von der Belastbarkeit[1] ist maximal auf Maschinenebene bedingt übertragbar. Das Abwechseln von ungeplanten Lastfällen, wie zum Beispiel tagelange „Nachtschichten“ begleitet von wochenlangen Phasen der Langeweile ist bezogen auf die Produktivität des Menschen negativ zu bewerten, man erinnere sich in diesem Zusammenhang an den bereits erörterten Bullwhip-Effekt.

Dieses Zusammenspiel stellt die eine Seite dar, die andere hingegen ist eher psychologischer Natur. Es gibt Menschen, die gerade bei abteilungsweiten Vorträgen, Meetings oder Veröffentlichung von Kennzahlen und Leistungsdaten im Vorfeld dieses Ereignisses bereits in eine Art psychosozialer Verspannung gelangen, die sie nur mehr teilweise klar urteilen lässt. Eine unsichere Datenbasis, beziehungsweise Darstellungsform, verstärkt natürlich das Stressmoment ungemein mehr.

5.2.7 Ressourcenexklusivität im Projektkontext

Das Modell d*er seriellen Bearbeitung* ist auch in Bezug auf die Softwareentwicklung nur schwer anwendbar. Nicht selten werden Ressourcen parallel benutzt. Es gibt die verschiedensten Motive dafür.

[1] Hierbei wird auch der Stressfaktor subsumiert

Einmal um das „alte“ Projekt noch sauber abzuschließen, oder komplexe Fehlersuche und Behebung in abgeschlossenen Projekten die bereits kommerzialisiert wurden und sich deswegen in der Wartungsphase befinden. Eine Entwicklergruppe wurde einem agilen Modell unterworfen und verfolgt die „Pair Programming“ Technik um Wissensverteilung zu forcieren, oder wurde an ein externes Projektstundenkontingent vermietet. In all diesen Fällen wird oft die Entwicklung irrigerweise mit einer einfachen Bedienung einer Maschine verwechselt, was jedoch nicht stimmt, da durch die heterogene Projektlandschaft bedingte *Einarbeitungszeit* oft übersehen wird.

5.2.8 Verfügbarkeiten diverser Unterstützungssysteme

Es stellt sich generell immer wieder die Frage nach einer adäquaten systemtheoretischen Unterstützung des Projektmanagements. Damit ist die Gegebenheit zwischen Supportsystemen die die Verfügbarkeit von Informationen bereitstellen einerseits, und Systeme die einen formalen Zugang im Projektwesen unterstützen andererseits gemeint.

Ersteres wird oft budgetärer Unzulänglichkeiten verwehrt, die zweite Gegebenheit aufgrund latenten Zeitmangels. Beide Argumente sind jedoch bei genauerer Betrachtungsweise unzulänglich, da sie nämlich beide aufgrund anderer Faktoren zustande kommen.

Größere Unternehmen setzen vermehrt auf kommerzielle Softwareprodukte, da es einerseits um die Sicherheit der Lizenzgebarungen geht, andererseits der angebotene Support attraktiv erscheint. Dieses Modell ist im Grunde genommen in Bezug auf Produktionssysteme plausibel, aber nicht notwendigerweise zwingend im Hinblick auf Supportsysteme des Projektmanagements. Seit der Ära der FOSS[1] Systeme gibt es durchaus zunehmend qualitativ hochwertige Produkte die meist den Anforderungen des regulären Projektmanagements genügen.

Ein weiterer Aspekt gilt der allzu oft praktizierten informellen Vermittlung von Informationen. Hierbei wird oft ein Zeitmangel bezüglich formaler Aufbereitung unterstellt. Auch diese Unterstellung ist jedoch nur bedingt gültig. Es gibt sicherlich abhängig von der konkreten Projektorganisation und den Ablaufstrukturen teilweise überlastete Kompetenzstellen, diese sollten aber eher die Ausnahme bilden, ansonsten müsste die Struktur an sich in Frage gestellt werden. Also der Großteil des Projektmanagements ist in der Regel nicht betroffen und bedient sich trotz allem hauptsächlich informeller Mittel.

Ein empirisch oft beobachteter Grund dafür, ist einfach die Bequemlichkeit und eine bewusst, durch eher unpräzisere Darstellungstechniken, bedingt herbeigeführte Unschärfe die dementsprechend auch mehr *Handlungsspielraum* offen lässt. Dieser Spielraum, dient vor allem den Projektbeteiligten als zeitlicher Puffer, um ihre individuellen Teilziele dennoch *quasi fristgerecht* zu erreichen, und um nicht aufgrund einer marginalen Projektplanzeitverzögerung eine sofortige Eskalation auszulösen.

[1] FOSS: Free Open Source Software

Falls man sich jedoch einfacher formaler semi-automatisch generierter Mittel bedienen könnte, wären beide Bedingungen dennoch erfüllt. Es wäre der Aspekt des minimalen Managementzeitaufwandes gegeben, wie auch ein gewünschter Handlungsspielraum, aber auf einer präziseren und eindeutigeren Datenbasis.

5.3 Das applikative Forschungskonzept

Aufbauend auf den zuvor angestellten Identifikationen bieten sich nun einige Lösungsansätze zur Problemreduktion an. Vor allem bedingt durch die große Schnittmenge der interdisziplinären Bereiche ergeben sich die verschiedensten Theorieansätze.

Es besteht die Möglichkeit im Kontext der Organisationslehre Adaptionen durchzuführen – im Bereich des Informationsaustausches, speziell der Prinzipal-Agent Theorie, Optimierungen zu erzielen – oder aber auch im Bereich der Unterstützungssysteme Ergänzungen zu machen. So könnte, angefangen von einem punktuellen theoretischen Lösungskonzept über methodologische Frameworks bis hin zu einer Softwarekomplettlösung, alles dem Zweck einer Lösungsfindung dienen.

Diese wissenschaftliche Abhandlung beschränkt sich in weiterer Folge auf einen *applikativen Ansatz*, um verstärkt eine Auswahl von Problemstellungen empirisch zu bewerten und abhängig von deren Ergebnissen, gegebenenfalls weitere theoretische Ansätze hinsichtlich einer Verallgemeinerung beziehungsweise Vertiefung des bestehenden pragmatischen Konzeptes zu formulieren.

Die Vorgehensweise wird als *applikativ* bezeichnet, da ein rein theoretischer Ansatz zur Untermauerung ungenügend wäre. Eine Applikation hingegen, die im Sinne eines „Proof of Concept“ anwendbar ist und aus dessen Resultaten Schlüsse gezogen werden können, ist bezüglich einer potenziellen Ausweitung des Konzeptes als ein *robustes Vehikel* anzusehen [KH2005]. Durch das in Kapitel 1.4 skizzierte Vorgehensmodell, stellt sie somit eine zentrale Instanz des Forschungskonzeptes dar.

Der Forscher nähert sich durch einen kollaborativ ausgetragenen „Design based Research“ Ansatz in iterativ inkrementeller Weise der Spezifikation einer Forschungssoftware, im Konkreten einem Software-Prototypen, um damit im Rückkehrschluss die aufgestellten Thesen zu testen. In und vor allem nach der Erprobungsphase der Implementierung in einem Testumfeld der Praxis, werden qualitative und quantitative Aspekte der Nutzer, sprich Experten, erhoben und in einer anschließenden Post-Analyse ausgewertet. Man ist dadurch in der Lage den Prototypen, angereichert durch die *ex post* Ergebnisse, stets zu optimieren und in beliebigem Detailgrad auszuarbeiten.

Eine wichtige Voraussetzung stellt vor allem die Skalierung der Unternehmens- und Testfeldgröße dar [MH2012]. Diese sollte in jeglicher Hinsicht variieren, da erwartet wird, dass die Ergebnisse auch von jenen Faktoren stark abhängig sind und das primäre Ziel eine Ausrichtung im Sinne einer Querschnittsbetrachtung hat. Die Erwartung setzt auf eine positive Korrelation der beiden Größen, sprich der Nutzen steigt mit der Komplexität der Organisationsstruktur sowie mit der

Komplexität des Projektes. Interessant sind vor allem Szenarien wo diese Korrelation nicht prognostiziert werden kann, wie zum Beispiel kleinere Teilprojekte in größere Unternehmen oder umgekehrt größere Projekte, speziell in Bezug auf die Komplexitätsordnung, in eher kleineren Unternehmungen.

Um diesen konkreten Apparat zu entwickeln und zu Nutzen bedarf es nun einiger Grundvoraussetzungen die im Weiteren diskutiert werden.

5.3.1 Problemselektion

Die Enumeration in Kapitel 5.1 stellt die sogenannte Potentialmenge der identifizierten Probleme dar. Da dieser Komplex jedoch den gesetzten Rahmen dieser wissenschaftlichen Erhebung sprengen würde beschränken wir uns auf eine wohl verstandene Auswahl der Probleme. Das zuvor beschriebene Forschungskonzept ermöglicht uns jedoch den Problemraum beliebig explorativ zu erforschen, da wir uns im kollaborativen Sinn jeglicher Problemstellung, bedingt durch die Offenheit der Vorgehensweise, widmen können.

Die erste Iteration über die Problemmenge, auf die sich die Arbeit stützt adressiert speziell die Problematiken

— effiziente Kommunikation

— Informationsverfügbarkeit

— Informationsasymmetrie

— Verfügbarkeiten diverser Unterstützungssysteme

— ex ante Abschätzungen des Softwareaufwandes

wobei, sofern sich auch signifikante Schlussfolgerungen ableiten lassen, andere zuvor aufgezeigte Problemstellungen sehr wohl im Zuge der Ergebnisauswertung erörtert werden.

5.3.2 Reformulierung der Problemstellung in der Praxis

Eine adäquate Reformulierung des Problems für den praktischen Ablauf in der „realen Welt“ ist unbedingt von Nöten. Falls das Problem nämlich zu abstrakt formuliert ist, kann sich der Teilnehmer nur schwer damit identifizieren und dadurch auch ungenügend motivieren. Das Vorhaben würde somit den eigentlichen Zweck, der kollaborativen Konsensbildung, verfehlen. Um die Kollaboration zu forcieren wird versucht, die Problemauswahl unternehmensspezifisch plakativ zu veranschaulichen ohne dabei bewusst einen Änderungsprozess zu veranlassen. Es wird somit jede beteiligte Unternehmung hinsichtlich ihrer Projektorganisationsstruktur und Ablaufprozesse im Vorfeld untersucht. Bei dieser Methodik etablieren sich zwangsweise Schlüsselfiguren die speziell im Hinblick auf Informationsengpässe eine wichtige Rolle spielen. Des Weiteren offenbaren sich auch indirekt die Infrastruktur und die Nutzungsintensität der Unterstützungssysteme, die wiederum eine durchgängige Darstellung bezüglich der Formalaspekte des Informationsaus-

tausches innerhalb der Projekthierarchie ermöglichen. Durch den Praxisbezug der involvierten Teilnehmer und durch den geforderten Dialog lassen sich durchaus interessante Verteilungsaspekte in Bezug auf die horizontale und vertikale Informationssymmetrie beobachten.

5.3.3 Abbildung des Lösungsraumes in eine mögliche Semantikebene

Ein Benutzergruppen übergreifendes gemeinsames Projektverständnis, ist die Voraussetzung für die Schaffung einer einheitlichen Begriffswelt. Es muss somit die *Zielsetzung und Bedeutung* der zu entwickelnden Notation klar und durchgängig verstanden werden.

Eine Herausforderung der besonderen Art stellt die gemeinschaftliche Entwicklung dar. Hierbei treffen zwei Fronten, die wirtschaftliche und technische, aneinander die sich vorerst nur zögerlich nähern kommen. Die Betrachtungsweisen und Intentionen divergieren teilweise beträchtlich. Die technische Fokussierung überlappt sich mit der wirtschaftlichen nur sehr begrenzt, im Gegenteil sie widersprechen sich sogar fallweise.

Verlangt eine technisch optimale Realisierung teilweise neue Mittel und Techniken, die das Projekt jedoch in den zeitlichen und monetären Grenzen sprengen würde, verfolgt das wirtschaftliche Trajekt hingegen einen optimalen Einsatz bereits vorhandener Ressourcen. Diese teils diametralen Anforderungen ergeben oft ein Konfliktpotential, dass über die gesamte Projektlebenszeit persistiert. Dieser Vergleich ist jedoch nur einer von vielen.

Ein weiteres Problem stellt die generell bewusste beziehungsweise unbewusste *Intransparenz* dar. Oft werden manch technische und wirtschaftliche Zusammenhänge nicht adäquat verstanden, da das benötigte Hintergrundwissen dazu fehlt, oder ein notwendiges Auseinandersetzen mit machen Details sehr häufig zu mühsam im Verhältnis zum zeitlichen Aufwand erscheint. Es gibt jedoch auch Anlässe wo bewusst die Transparenz seitens der Teams verhindert wird, da es im Rückblick auf eine zum Beispiel bereits gelieferte Softwareabschätzung stark im zeitlichen Hintertreffen ist, und den Zeitverlust noch kompensieren möchte.

Beide Verhaltensweisen sind aus theoretischer Sicht nicht wirklich tragbar. Das Argument des fehlenden Hintergrundwissens könnte man leicht mit spontanen Nachfragen oder im Vorfeld mittels geplanter interner Workshops zwecks Wissensaustausch ergänzen, wie es bei agilen Vorgehensmodellen sogar gefördert wird [PP2014b]. Die Notwendigkeit des bewussten intransparenten Verhaltens könnte ebenfalls in gewissen Bereichen hinfällig werden, wenn man einerseits die Möglichkeit und Mittel einer zeitnahen Entdeckung solcher außerplanmässigen Projektsituierungen hat und verlässliche Kommunikationsstrukturen zur adäquaten Eskalation zur Verfügung stehen.

5.3.4 Kooperatives Entwickeln einer einheitlichen Notation

Falls nun die Unwegsamkeiten einer gemeinsamen Semantik geebnet sind und man sich darüber verständigt hat bedarf es nun einer, über alle involvierten Bereiche, einprägsamen Syntax. Diese Notation sollte zumindest den folgenden Forderungen genügen.

— Orthogonalität & Eindeutigkeit

— Erweiterbarkeit

— Einheitlichkeit

Die erste Forderung verlangt, dass die Bedeutung der syntaktischen Schlüsselworte einer Schnittmenge *disjunkter* Elemente gleichkommt – es wird also keine Überlappung der Semantik gestattet. Sie impliziert überdies eine eineindeutige (bijektive) Auszeichnung der Semantik, indem eine Begrifflichkeit ausschliesslich mit einem eindeutigen Schlüsselwort logisch verbunden ist. Die zweite Bedingung hingegen setzt voraus, dass die Formulierung eines gewünschten Sachverhalts mit einer bestimmten Kombination von Schlüsselwörtern erzielbar ist [LK2012a]. Die Einheitlichkeit an sich, garantiert dass die Verwendung definierter Strukturen in Bezug auf die Schlüsselwortmenge stets gleiche Bedeutung und Aussagekraft besitzen.

Diese drei Bedingungen klingen anfänglich implizit und trivial, bei genauerer Betrachtung hingegen wird schnell klar, dass diese Forderungen stark *kontextabhängig* sind. Man stelle sich die Forderung nach Erweiterbarkeit vor, in dem man zuerst die semantischen Aussagen definiert und danach diese Fragmentierung in einen minimal programmtechnisch grammatikalisch Satz überführt.

Ein einfaches Beispiel soll der Erläuterung dienen:

Falls die Version einer Software in Bezug auf ihre Abhängigkeiten und Vollständigkeit gemäß einer zeitlich begrenzten Feature-Roadmap dargestellt werden soll, benötigt man die Fragmente der Versionsinformation, der Abhängigkeitsinformation anderer Module und den derzeitigen Status der Bearbeitung in zeitlicher Manier.

Es sind somit 4 Artefakte, wobei 3 explizit benutzt werden. Eine logische Konjunktion ergibt letztendlich den Sachverhalt wieder:

Version + Abhängigkeit + Bearbeitungsstatus = „die Version einer Software in Bezug auf ihre anderen Abhängigkeiten sowie Vollständigkeit und Bearbeitungsstatus gemäss zeitlich begrenztem Intervall.“

Der Bandbreite der möglichen Syntaxformen ist groß. Der Bogen spannt sich von komplexen DSL[1] bis hin zu einfachen atomaren Schlüsselwortfolgen. Eine adäquate Wahl hängt von verschiedenen Faktoren ab. Ein wichtiges Faktum ist sicherlich generell die Komplexitätsklasse in der die Applikation ihre Anwendung findet, jedoch nicht nur dies – ein weiterer entscheidender Beweggrund ist vor allem

[1] Domain Specific Language

das involvierte Publikum, sprich die Menschen, die mit diesen Idiom operieren müssen.

Die Frage nach einer DSL wird in der Regel durch die Notwendigkeit einer eigenen Abbildung einer „Wissensdomäne“ beantwortet. Oft genügen einfachere Ansätze den Vorhaben, denn der Aufwand der Erstellung eigener Grammatiken sollte beim Entwurf einer benutzerzentrierten Sprache unbedingt dem praktischen Einsatznutzen gegenübergestellt werden. Meist geht hier die Praxis den einfacheren Weg und nutzt nur ungern syntaktische Raffinessen, da alle Beteiligten dasselbe Verständnis bezüglich der Syntax aufbringen und auch anwenden müssen. Es ist notwendig, dass die Vielzahl der im Zuge der Syntaxdefinition geforderten Parameter einerseits vereinheitlicht und andererseits auch auf ein Minimum beschränkt werden, um einer unbewussten mehrdeutigen Interpretationsmöglichkeit entgegenzuwirken. Das System würde sich schnell bezüglich eines positiven Nutzens *ad absurdum* führen, da eine Mehrdeutigkeit einer Nichtbeantwortung einer Frage gleichkäme, ähnlich einem gleichzeitigen „Ja“ und „Nein“ auf eine Entscheidungsfrage.

Natürlich muss die definierte Syntax auch den Anforderungen des darauf technisch basierten Systems, das hier als Wirt genutzt wird, vor allem bezüglich benutzter Zeichen des verwendeten Zeichensatzes und der verbotenen Terminalsymbole und Schlüsselwörter genügen. Um einer potenziellen *multilingualen* Nutzung des Systems entgegenzukommen ist es ratsam sich bei der Neudefinition einer Syntax *nicht ausschließlich* an den einfachen ASCII-Zeichensatz zu binden, der zwar einerseits als „größter gemeinsamer Nenner“ aller Wirtssysteme zu einander kompatibel ist, aber andererseits sprachliche Eigenheiten wie Umlaute nicht berücksichtigt.

5.3.5 Technologieauswahl

Der nächste kategorische Schritt hinsichtlich Realisierung des Softwaretools stellt sicherlich die Technologieauswahl dar, hier im Konkreten der Technologieeinsatz zur Softwareimplementation. Durch diese Festlegung sind in gewisser Hinsicht auch bereits Bahnen und Schranken vorgegeben. Ermöglichen bestimmte Aspekte der Implementationssprache bessere Modellierbarkeit der realen Welt, so können auch Anbindungen an Drittsysteme diesbezüglich Schranken der Einsetzbarkeit setzen.

Sind es sogenannte „single purpose“ Anwendungen, solche die ausschließlich auf einer bestimmten Architektur laufen sollen beziehungsweise müssen oder keine breite Einsatzfähigkeit besitzen, also speziell sind, oder eher breitere Anwendungen, die ein großes Publikum beziehungsweise bedingt durch ihren Einsatz in ein potenziell hybrides Betriebssystem- und Anwendungsumfeld haben.

Verwendet man etwa Implementationssprachen die auf einer virtuellen Maschine laufen, wie zum Beispiel Java und .NET, wird implizit eine hohe Portabilität gewährleistet. Die Zielapplikation ist somit, bezüglich Prozessorkern und Betriebssystem, größtenteils plattformunabhängig, auch Subsysteme wie ein grafisches

Framework zur GUI-Modellierung werden meist als integraler Bestandteil des Laufzeitsystems mitgeliefert. Als Nachteil, wären spezielle Anforderungen in Bereichen der Performance sowie die der Applikationsgröße anzusetzen, wobei hier der Grad der Optimierung der VM[1] in den letzten Jahren stark zugenommen hat und diese Argumente nur mehr bedingt Gültigkeit besitzen.

Im Fall einer hohen Anforderung an die Rechenleistung hingegen, werden oft *native*, also kompilierte Sprachen, die direkt in optimierten Maschinencode der Zielarchitektur übersetzt werden, angewandt. Einige Vertreter dieser Gattung wären C/C++ und Ada. Alle haben jedoch den Nachteil meist erhöhter Portabilitätsprobleme und auch bedingt hoher Einarbeitungszeit in die Sprache selbst.

In letzter Zeit kommen speziell Programmiersprachen zur Anwendung die interpretativer Natur sind. Solche interpretierte Sprachen wären zum Beispiel Python, Ruby und REBOL[2]. Diese Sprachfamilie besitzt zumeist die Vorteile, bedingt durch eine interpretative Konsole, kurz REPL[3], des schnelleren Prototypings sowie einer Spezialisierung hinsichtlich *interner DSL*, einer Möglichkeit Wissensdomänen mithilfe selbstdefinierter Idiome basierend auf dem Sprachkern programmatisch abzubilden.

Exkurs: interne und externe DSL

Um diese Unterscheidung besser zu verstehen muss zuerst eine klare Abgrenzung zu den gewohnten Programmiersprachen gezogen werden. In der Computerwissenschaft bedeutet das eine allgemein anwendbare Sprache, eine *general purpose language* (GPL), im Gegensatz zu einer Domänen spezialisierten Sprache, kurz DSL, Turing Vollständigkeit besitzen muss. Vereinfacht ausgedrückt bedeutet das, dass jeder berechenbare und somit auch entscheid- und aufzählbare Algorithmus implementierbar sein muss. Es kann prinzipiell zwischen zwei Arten der DSL Generierung unterschieden werden.

Interne DSLs werden meistens in Form von monolithischen Bibliotheken basierend auf einer darunterliegenden Wirtsprache angeboten. Es werden häufig dazu reflexive[4] (*meta programming)* Sprachmerkmale genutzt. Die Erstellung einer internen DSL ist verhältnismäßig relativ einfach, da sie sich auf die gewohnten Sprachmerkmale reduziert. Klassische Vertreter dazu wären *Ruby on Rails* oder REBOL.

Externe DSLs hingegen werden mittels separater Toolchains gebildet. Sie stellen somit ein eigenes Programm mit einer optimal entworfenen Syntax zur Domänenbeschreibung dar. Eines der bekanntesten Werkzeuge dieser Gattung ist sicherlich ANTLR[5], ein sogenannter Sprachgenerator der eine Mischung aus formaler Grammatik und objektorientierter Sprachelemente zur Implementation anbietet. Die

[1] VM: Virtuelle Maschine

[2] REBOL: Relative Expression Based Object Language (http://www.rebol.com)

[3] ReadEvalPrintLoop: Konsolenfenster mit Interpreter-Prompt

[4] In der Fachliteratur wird häufig der Begriff meta programming genutzt

[5] ANother Tool for Language Recognition

DSL Generation ist dadurch auch weitgehend Sprach- und Plattformunabhängig [TP2009].

Damit ergeben sich die unterschiedlichsten Ausprägungen bezüglich der Vor und Nachteile. Die Einarbeitung und die Benutzung interner DSLs ist wesentlich einfacher vorrausgesetzt die zugrundeliegende Wirtssprache wird beherrscht. Hinsichtlich der Performance wäre bei großen Domänenbeschreibungen eine extern generierte DSL vorzuziehen.

Ebenso im Fall der Fehlerausgaben kann das externe System wesentlich gezieltere Angaben machen, wobei die interne DSL einen Fehler meist nur in Form eines *Stacktraces*[1] quittiert. Die folgende Abbildung 5.1 stellt die beiden Sprachformen unter Berücksichtigung des Einsatzzweckes gegenüber.

DSL	**GPL**
textuelles / grafisches Modell	Quellcode
Zweck intuitiv besser lesbar	Zweck erfordert genauere Analyse des Quellcodes
Wissen aus Domänen	Wissen basierend auf Von-Neumann Architektur
abhängig von Domäne einsetzbar	unabhängig von Domäne einsetzbar
Anpassung erfordert Modelländerung (Semantik einfacher prüfbar)	Anpassung erfordert Quellcodeänderungen / Gefahr, dass Quellcode 'unüberschaubar' wird
DSL-Programmierer konzentrieren sich direkt auf Domänenproblem mit Hilfe der Domänensprache / des Domänenwissens	GPL-Programmierer konzentrieren sich auf Domänenproblem (Verständnis vorhanden?) und auf die komplexe Quellcodeumsetzung gleichzeitig
Domänenkonstrukte für detaillierte Problembeschreibung	Sammlung von Methoden/Funktion, direktes Mapping von Domänenkonstrukten auf APIs schwer oder gar nicht möglich

Abbildung 5.1 – DSL versus GPL ein Vergleich

5.3.6 Implementierung einer Kernfunktionalität

Dieser Teil der Softwarekomponente widerspiegelt einen zentralen Teil der Applikation. Wie in *Teil III* noch näher erläutert werden wird, gibt es abhängig von den Anwendungserfordernissen gewisse applikative „hot spots[2]". Diese Stellen bedürfen einer sorgfältigen Analyse im Vorfeld, da sie in gewisser Weise auch das Rückgrat der Applikation symbolisieren und ein unausgewogener Entwurf die Applikation stark in Mitleidenschaft ziehen würde. So sind Faktoren wie die Reaktionsfreudigkeit einer Benutzschnittstelle (Responsiveness) oder auch Spei-

[1] Diagnostische Ausgabe und Interpretation des Inhalts eines Programmstapels
[2] http://en.wikipedia.org/wiki/Hot_spot_(computer_programming)

cherauslastungen meist signifikante Indikatoren bezüglich der Implementierungssorgfalt jener Kernfunktionalitäten.

Eine etwaige Anbindung an Drittsysteme verkompliziert die Sichtweise, da man den Verhaltenseigenschaften und Anforderungen dieses Subsystems genügen muss, da es ansonsten während des Betriebes womöglich zu unvorhergesehenen Systemausfällen kommen kann, oder noch schlimmer unkorrelierte Datenfragmente erhält, die aus logischer Sicht keinen Sinn ergeben aber aus der applikativen Perspektive dennoch verarbeitbar sind.

Häufig fehlen dem Verarbeitungskern auch Ressourcen wie Prozessorzeit da einfach synchrone, anstatt paralleler, Abarbeitung oder nicht adäquate Speicherstrukturen, wie Listen statt Streams zur Datenhaltung, zum Einsatz kommen. Auch Inkonsistenzen in der Verarbeitung sollten konsequent behandelt werden um einen Systemausfall zu verhindern, sofern die Weiterausführung überhaupt noch sinnvoll ist, ansonsten mit einer eindeutigen Fehlersystematik korrekt terminieren. Die Softwaresystemarchitektur ist der Treiber in Bezug auf die Kapselung und Nutzung des Kerns aus der Sicht der diversen Schnittstellen. Es gibt diesbezüglich mehrere Möglichkeiten zur Implementierung des Verarbeitungskerns, sei es ein eigener abgeschotteter asynchron kommunizierender Prozess, POSIX konforme Interprozesskommunikation auf Betriebssystemebene oder auch nur einige Funktionsprimitive die in gesicherter Weise koordiniert und atomar ablaufen.

Diese Problemstellungen und Bottlenecks[1] ergeben sich vor allem nach intensiver Befassung und Einarbeitungszeit an das anzukoppelnde System und werden häufig erst abhängig von echten Praxisdaten sichtbar, da zumeist simulierte Testdaten einen weit kleineren Umfang besitzen. Wie sich bereits in einer Voranalyse zeigte, war es notwendig, Voraussetzungen für die zu erwartenden Verarbeitungsanforderungen abzuschätzen, und dementsprechend adäquat zu dimensionieren um einen skalierenden Mechanismus der Forschungssoftware RBMST erst entwerfen zu können.

5.3.7 Kooperativer Entwurf der Benutzerschnittstellen

Ein System wird nur dann regelmäßig angewandt, sofern durch den Benutzer die Eingabe der benötigten Daten auch in ergonomischer Hinsicht geschehen kann. Der Ergonomiefaktor wird auch diesbezüglich oft unterschätzt. Eine komparative Studie hat ergeben, dass ein hoher Prozentsatz der Arbeitszeit durch unergonomische Schnittstellen unproduktiv verstreicht. Insbesondere Forscher der *Fachhochschule für Technik und Wirtschaft in Berlin* gehen dieser Frage nach und identifizierten drei wesentliche Aspekte der *Usability*[2].

— Prozessgestaltung

— Produktivität

— Arbeitstressverringerung

[1] Bottleneck: Flaschenhals, entspricht salopp einem Ressourcenengpass

[2] http://de.wikipedia.org/wiki/Gebrauchstauglichkeit_(Produkt)

Zur Software-Ergonomie, ein Synonym der Usability, lässt sich noch hinzufügen, dass gemäß *ISO 9241* „Ergonomie der Mensch-System-Interaktion“ Anforderungen an den „Dialog“ definiert sind, die somit auch einen Verbindlichkeitscharakter darstellen.

Der Aspekt der Prozessgestaltung ist sicherlich der Wesentlichste. Sehr oft werden Interaktionsschnittstellen konzipiert die im engeren Sinne, oft unbewusst, einen Prozessablauf, und somit einen personellen Arbeitsablauf vorgeben. Im negativen Fall ist es ein erzwungenes Anpassen eines betrieblichen Ablaufes, welches sich dann meist auf die Produktivität kontraproduktiv auswirkt. Eine Vielzahl möglicher Schnittstellen zur Dateneingabe schafft dieser Problematik oft Abhilfe, da dadurch etliche, auch unbedachte, Einsatzfälle abgedeckt werden.

Die Produktivität hingegen hängt nicht nur von dem vorgegebenen Workflow ab, sondern auch von einem adäquaten „Look & Feel“ der Applikation. Werden Inhalte zu klein, Eingabemasken zu kurz oder die Menüführung zu kompliziert aufgebaut – alles übt Einfluss auf die Gesamtproduktivität aus. Dieser „verlorene Zeitfaktor“ während der Eingabe hängt natürlich auch stark von der Frequenz der Bedienung ab.

Angenommen wenn, wie häufig bei semi automatisierten Tests, Testdaten geändert werden müssen und die Eingabe sich jeweils um 3 Sekunden pro Änderung durch eine zu kleine Eingabemaske verzögert, wären bei einem Serientest von 2000 Stück bereits 9000 Sekunden, also mehr als 2 Stunden und 40 Minuten an unproduktiver Zeit verloren gegangen.

Der Faktor Stress ist sehr oft eine implizite Folge der zuvor beschriebenen Projektverzögerungen, kann jedoch auch explizit auf triviale Fehler in der Software zurückgeführt werden. Einfache Fehler wie zum Beispiel eine notwendige Doppeleingabe eines Datum oder nicht abgefangene Fehleingaben die das Programm abstürzen lassen führen leicht zu latenten Stressphasen die sich langzeitlich negativ auf die Gesundheit der Beteiligten auswirkt. Eine logische Folge dieser Einflüsse sind häufig psychosomatisch bedingte Ausfälle der Arbeitskräfte [AA2004] – das angestrebte Projektziel verzögert sich nun noch mehr.

All jene Faktoren waren bis in die Mitte der 1990’er Jahre noch weitgehend unerforscht und wurden durch den vermehrten Einsatz webbasierter Anwendungen zunehmend evidenter.

5.3.8 Evaluierung eines adäquaten Testumfeldes und Zeitraumes

Die Referenz-Implementation soll vor allem in einem Praxisumfeld ihre Anwendbarkeit unter Beweis stellen. Hierzu muss jedoch der projektierte Einsatzrahmen festgestellt werden. Einerseits sollen die Testdaten tatsächlich „real world“ Charakter besitzen, andererseits muss, um die Applikation anwendbar zu machen, der Workflow der„commit Phase[1]“ von Software Artefakten teilweise ergänzt werden.

Dazu bedarf es grundsätzlich

[1] Vorgang indem Dateien in ein Repositorium ordentlich eingepflegt werden

— Projektauswahl im Einklang mit der Unternehmensgröße

— Einschulung der Operatoren

— Vereinbaren eines „sinnvollen“ Testzeitrahmens

— Akkordieren von Projektstatusmeetings beziehungsweise auch Audits

Um eine große Bandbreite bezüglich der Effektivität der Applikation untersuchen so können, müssen im Vorfeld die Unternehmensgrößen im Einklang mit den dort aktiven Projekten analysiert werden. So besitzen meist größere Unternehmen eine andere Unternehmensstruktur und auch Kultur als im Gegensatz zu KMU[1]. Diese Diversität stellt gerade die Herausforderung bezüglich eines optimalen Testfeldes dar. So bedarf es eines bestimmten Versionskontrollsystems wie auch einem adäquaten Workflow in der Projektdurchführung. Zyklische Meilensteinterminisierung oder ähnliche Statusfeststellungen sind von gewisser Notwendigkeit um den Output des Tools messen zu können. Um eine unbegründete Angst im Sinne einer totalen Transparenz zu nehmen, sollten die Vereinbarungen stets im Kollektiv, wo das klassische Management mit der Technik vorerst harmonieren muss, getroffen werden. Es gilt somit diese benötigten Variablen in Einklang zu bringen, da anderenfalls das Evaluierte den prognostizierten Zweck verfehlen könnte. Ein zeitlicher Spielraum betreffend Anpassungen der prototypischen Software an das konkrete Repository des darunterliegenden Versionssystem sollte jedenfalls mit einkalkuliert werden, da das Programm zuvor noch keinen echten praktischen Einsatzfall erfahren hat.

5.3.9 Auswertung der gesammelten Daten

Kernstück der zuvor exerzierten Erprobungsphase stellt die gewonnene Rohdatenbasis dar. Basierend auf diesem Stamm werden durch angewandte statistische Analyseverfahren Schlüsse und Analogien in pragmatischer Weise gezogen, die die wahre Sinnhaftigkeit zur weiteren Verfolgung des Ansatzes begründen oder verwerfen. Um eine vollständiges Spektrum der möglichen Evaluationsparameter zu erhalten, werden drei getrennte Betrachtungsweisen genutzt.

Die erste Betrachtung stützt sich durch die Fragebogentechnik auf ein vorwiegend statisch quantitatives Ergebnis in Bezug auf eine etwaige Zeitersparnis in der Informationsbeschaffung sei es gegeben durch die ständige Verfügbarkeit der benötigten Information oder auch einer Minimierung der Aktualisierung von Metriken. Es sind auch Sekundärinformationen wie beispielsweise ein dimensionsloser Kostenfaktor, der der Gewichtung diverser Abschätzungen dienen kann von Interesse, vor allem in Bezug auf die Projektorganisation und Projektkomplexität. Bei all diesen erfassten Werten werden einfache statistische Lagemaße verwendet um die Nachvollziehbarkeit nicht zu behindern. Im Bedarfsfall können leicht darauf basierend erweiterte Statistikfunktionen angewandt werden.

Die zweite Betrachtung widmet sich mehr der qualitativen Bewertung. Hierbei werden mittels selektiver Experteninterviews das subjektive Empfinden hinsichtlich

[1] Kleinere und mittlere Unternehmen

Bedienbarkeit, emotionaler Gegebenheiten wie auch zukünftiger Engagements hinterfragt. Den dritten Teil stellt die stetige Beobachtung und Sammlung der Erfahrungswerte aus der direkten Feldforschung dar. Hierbei werden vorrangig kollektive Verhaltensmuster und Tendenzen studiert. Wichtig hierbei ist vor allem, dass die Befragungen und anschließenden Auswertungen in anonymer Weise stattzufinden haben, da die Mehrheit der Beteiligten doch nicht in direktem Zusammenhang mit ihren persönlichen Empfindungen und Angaben gebracht werden wollen.

5.3.9.a Nutzenauswertung

Die vorher aufgestellten Prozeduren dienen ausschließlich dem Zweck einer gewichteten harmonisierten Betrachtung der erfassten Werte. Es wird der Versuch unternommen einen „Quasinutzen“ zu eruieren und diesen im Kontext der Unternehmensgröße wie auch Projektkomplexität abzubilden. Der Nutzen setzt sich vorwiegend durch eine gewogene Querschnittsbetrachtung aller Facetten zusammen. Es wird auch erwartet, dass sich Teile der unternehmerischen Vorgehenskultur in subtiler Weise verändern. Diese Vorgänge, wie ein leicht geänderter Workflow oder aber auch emotionale Einflüsse wie Stress, fließen in die nummerische Nutzenbewertung zwar nicht direkt ein, werden aber mittelbar durch ausgewählte Befragungen und Beobachtungen erfasst.

In diesem Zusammenhang bieten sich viele Vorgehensweisen bezüglich einer Nutzenermittlung an. Der Vorteil dieser Analyse jedoch ist die *relative Stabilität* ihrer Aussage hinsichtlich der Skalierbarkeit der beiden Größen, Unternehmen und Projektkomplexität. Da die Daten stets in unverfälschter Rohform samt der angewandten Analyseverfahren parat liegen, kann jedes erdenkliche Nutzenkalkül, sofern es auf die Charakteristik der Daten anwendbar ist, aufgestellt werden.

5.3.10 Evaluation und iterative Konzeptverfeinerung

In der wissenschaftlichen Manier mit den gewonnen Daten das angewandte Konzept zu verbreitern, beziehungsweise bereits analysierte Domänen im Verständnis zu vertiefen, konzentriert sich die Bildung eines Konnex. Basierend auf diesen Erkenntnissen können Annahmen verworfen werden oder auch neue Konzepte entstehen. In jedem Fall sollte die Konzeptverfeinerung im gewohnten Trajekt, sprich im kollaborativen Kontext, stattfinden da ansonsten das Modell mit seinen vorher abgeleiteten Methoden und Erkenntnissen nicht mehr schlüssig wäre. Ein individueller Ansatz, losgelöst von jeglichem unternehmensweiten Zuspruch, würde somit die gestellten Anforderungen nicht erfüllen. Im Gegenteil, ein optionale Vergrößerung des beteiligten Publikums könnte vielmehr in Erwägung gezogen werden, um auch etwaige weitreichendere Handlungen ins Kalkül ziehen zu können.

Dieser Ansatz würde jedoch den derzeitigen Rahmen, nämlich die ausschließliche Fokussierung auf das Versionskontrollsystem als Informationsbasis sprengen, da es hierzu mehrerer Supportsysteme bedarf um eine strategische Handlung im Kontext der Unternehmen rational zu begründen.

6 Kapitel

Die These und ihre Anforderungen

Nicht weil es schwer ist, wagen wir es nicht,
sondern weil wir es nicht wagen, ist es schwer.
Seneca

Schwerpunkte in diesem Kapitel:

- Repository Based Management Support System
- Die Notwendigkeit einer eigenen Ontologiesprache
- Das RBMSS Vokabular
- Typisierung des Namensraumes von RBMSS

6.1 Repository Based Management Support System

In der Tradition der Wirtschaftsinformatik angeknüpft, wurde ausgehend von den notwendigen Grundlagen zum generellen Verständnis von Informationssystemen und über deren konkrete Einsatzweise im IT-Projektmanagement das benötigte Wissen kanonisch aufgebaut.

Die identifizierten Problemstellungen ergaben eine holistische Sicht über die ineinandergreifenden Disziplinen. Basierend auf diesem Themenkomplex wurde eine differenzierte Methodologie aufgebaut, um sich bestimmter Problemkreise zu nähern und unter der Berücksichtigung aller beteiligten Faktoren zu analysieren. Diese Vorgehensweise lehnt sich etwas an die „experimentelle Forschung“ an, da auch hier eine oder einige Variablen, gemäß dem Prinzip „*cetris paribus*“, eines bestehenden Systems geändert werden um eine Reaktion, ein geändertes Systemverhalten feststellen zu können. Die Feststellung erfolgt oft mit der Beobachtung ob die Hypothesen gar nicht, positiv oder negativ korrelieren.

Den hier verfolgten Ansatz kann man nun folgendermaßen beschreiben:

Es soll ein Software-Werkzeug, in weiterer Folge als RBMST bezeichnet, erstellt werden, dass in prototypischer Hinsicht einen „proof of concept“ zu den aufgestellten Hypothesen wie Verbesserungen in der Informationsaktualität und der Informationsrepräsentation leisten soll. Der Ansatz dabei ist, dass das Modell kollaborativ entworfen werden soll und anschließend in der Kohorte auch zur Anwendung kommt. Diese Forschungssoftware konzentriert sich auf ausgewählte Aspekte des Problemkreises und beleuchtet in der Erprobungsphase deren Verän-

derungen. Aufgrund dieser Beobachtungen werden Analogien zu verwandten Problemstellungen gezogen und versucht diese, in iterativer Weise, durch das Tool mit zu behandeln.

Dieses systematische Vorhaben wird von nun an unter den Titel *Repository Based Management Support System,* kurz RBMSS, subsumiert da dieses Akronym, meiner Meinung nach, in kurzen Worten die Kerncharakteristik prägt. Die kollaborative Erhebung der diversen Anforderungen stellt, im wissenschaftlichen Sinne, die eigentliche Leistung dar. Die gemeinsame Konsensbildung über einen Diskurs kommt häufig im Design stark verteilter Systeme zum Einsatz, um die komplexen Verschränkungen vollständig erfassen zu können. Ein Zitat zu kollaborativer Softwareentwicklung soll das Bild komplettieren [KT2012]:

„Neue Herausforderungen aufgrund der gestiegenen Komplexität und Dynamik bei der Herstellung von Produkten oder der Durchführung von Dienstleistungen erfordern die Zusammenarbeit vieler Beteiligter. Zusätzlich nimmt der Bedarf an Unterstützung einer solchen Zusammenarbeit aufgrund der sozialen Veränderungen im Arbeitsumfeld wie Globalisierung, Flexibilisierung und Innovationsdruck sowie der Entwicklung neuer Organisationsformen wie Tele-Arbeit, Offshoring und Outsourcing immer mehr zu. Darüber hinaus wird durch die Vernetzung von Computern, insbesondere durch das Internet, die Nachfrage nach einer integrierten Unterstützung der Zusammenarbeit stetig erhöht. Der Begriff Kollaboration wird oft sehr unterschiedlich verwendet. Er steht meistens nicht für sich alleine, sondern wird oft mit anderen – ähnlichen – Begriffen wie Kooperation, Koordination und Kommunikation genannt. Die Vorsilbe „Ko“ drückt eine Beziehung zwischen zwei Subjekten oder Objekten aus, die „miteinander“, „zusammen“ oder „gemeinsam“ handeln.“

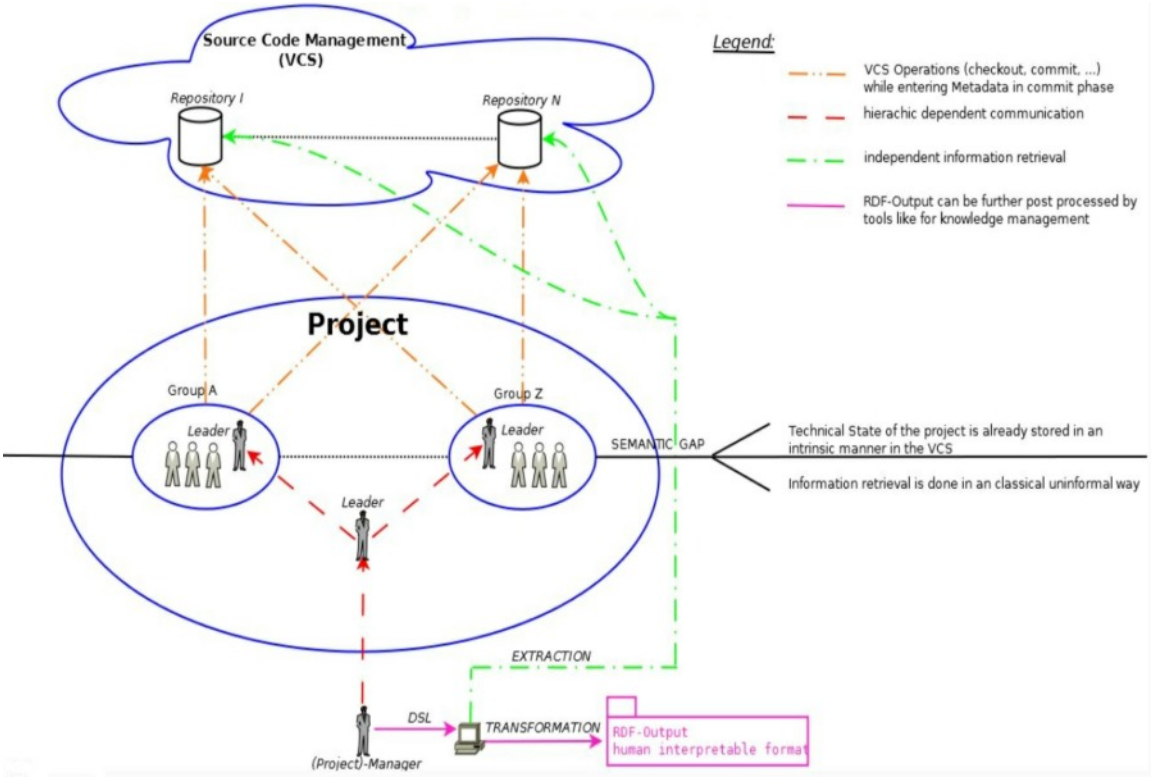


Abbildung 6.1 – Repository Based Management Support

Ein System welches basierend auf bestehenden Informationen eines projektbezogenen Repositoriums, das Management mit zusätzlich visualisiert aufbereiteter Information als Unterstützung in der Entscheidungsfindung dient. Entscheidend ist hierbei, dass das Konzept keine speziellen Ansprüche an die verwendbaren Repositorien stellt, lediglich ein dynamischer Projektverlauf muss manifestiert sein. Somit steht der Einsatz auch anderen Instanzen, die nicht Versionskontrollsystem Kennzeichnung besitzen, ebenfalls offen. Eine konkrete Systemimplementierung, also eine konkrete RBMSS-Instanz, ist die Forschungssoftware RBMST, ein Akronym für *Repository Based Management Support Tool.* Im weiteren Verlauf dieser Arbeit werden, abhängig von der Zielsetzung der Diskussion, manchmal die Ausdrücke „Proof of Concept", SW-Prototyp, Tool, Forschungssoftware und RBMST synonym verwendet.

Die Abbildung 6.1 zeigt prinzipiell mit ihren orangenen und roten strichpunktierten Linien eine klassische Ablaufhierarchie im Bereich der IT-Projektorganisation auf. Der rote Pfad stellt einen sogenannten kritischen Pfad dar. Eine Verbindung die sehr *fragil* ist. Darunter wird verstanden, dass im Ausfallsfall zum Beispiel krankheitsbedingt wie auch im Fall einer Informationsüberflutung oder unzureichender Kompetenz, eine optimale Entscheidungsgrundlage dem Projektmanagement nicht zur Verfügung gestellt werden kann, da der vereinbarte Informationsfluss in Qualität und Zeit nicht mehr gegeben ist.

Der Zweitpfad, auch *Backup Pfad*, mit den grünen strichpunktierten Linien koppelt sich wie der erste Pfad an die gleichen Quellen, den Versionskontrollsystemen der Softwareentwicklung, und endet auch beim Projektmanagement, jedoch in einem Informationssystem. Er stellt somit eine Art Redundanz dar, die aber im dynamischen Kontext des Managements als Backup- oder Ergänzungssystem betrachtet werden kann. Somit besitzt das System vielseitige Einsatzmöglichkeiten, wie die erhöhte Verfügbarkeit, bessere Informationsqualität, schnellerer Zugriff durch ad hoc Informationsextraktion sowie gleich aufbereitete Informationsdarstellungen. Die Abbildung suggeriert ebenfalls, eine offensichtlich positive Korrelation des Nutzens des Tools mit der Projektorganisationsgröße sowie der Projektkomplexitätsverteilung.

Diese Informationsquellen werden durch eine theoretisch unbegrenzte Anzahl an Repositorien dargestellt, worin sich unternehmenspezifische Informationen in fragmentierter Form befinden. Die Extraktion und Aufbereitung solcher Informationen, ermöglicht die Explizierung neuen Wissens, welches gemäß des *Wissenswürfels*[1] die primäre Form eines *internen impliziten kollektiven Wissens* besitzt. Intern deshalb, da es unternehmenspezifisches Wissen darstellt. Implizit deswegen weil es in keiner Form dokumentiert ist. Kollektiv dadurch, da dieses Wissen einer Gruppe und nicht einem einzigen Individuum zuzuordnen ist und auch die Verfügbarkeit lediglich im Kollektiv verbleibt. Um nun diese Informationen in semantischer Weise nutzen zu können müssen einerseits Ontologien verbreitert werden, und andererseits die Darstellungsform auch der Erwartungshaltung der Person, die pragmatisches Wissen ableiten möchte, adäquat angepasst werden.

[1] Günter Warnecke et al. 1998: Referenzmodell Wissenmanagement

6.1.1 Lingua Franca

Wie schon im alten Rom, bedient man sich hierbei am besten einer *gemeinsamen Handels- beziehungsweise Verkehrssprache*. Die Frage der Sprachtiefe ist stark abhängig von den darzustellenden Sachverhalten. Es bildet auch das Publikum, sprich die potenziellen Sprachanwender, einen entscheidenden Faktor. So ist eine kohärente Gruppe die in einer homogenen Begriffswelt beheimatet ist, etwas anderes als wenn mehrere Interessensgemeinschaften über ein und dasselbe Faktum diskutieren und entscheiden müssen.

Es stellt sich somit die Frage nach einer übergeordneten Begriffswelt, einer Ontologie, die den *gemeinsamen Nenner* der involvierten Gruppen formal abstrahiert und dokumentiert, sowie auch Erweiterungsmöglichkeiten in Abhängigkeit der Anforderungen der Sprache zulässt. Ob nun die konkreten Anforderungen einer eigenständigen Ontologiedefinition bedürfen, gilt zu klären. Folgende Aspekte werden dabei in Betracht gezogen

— Notwendigkeitsanforderung einer Ontologie

— Einsatzbereitschaft zum Erlernen neuer Sprachmittel

— Aufwand zu Substitutionslösungen abwägen

6.2 Die Notwendigkeit einer eigenen Ontologiesprache

Die Entscheidung einer eigenständigen Ontologiedefinition hängt stark vom Einsatzfall ab. Inwiefern soll diese Begrifflichkeit weiterverarbeitet werden, maschinell, oder dient sie nur einer Visualisierung ? Gibt es sehr viele semantische Abhängigkeiten die es zu Modellieren gibt, oder besitzen die Verbindungen eher nur den Charakter einer schlichten Aussage ?

In der Fachliteratur gibt es viele Abhandlungen zu dieser Fragestellung. Zuerst sollen ein paar gleichmächtige Ontologie Definitionen paraphrasiert werden. Eine der am häufigsten verwendeten Definitionen ist sicherlich die von *Studer* [SR2008].

„Eine Ontologie ist eine formale, explizite Spezifikation eines gemeinsamen Verständnisses einer Begrifflichkeit."

Formal bedeutet die Forderung nach einer „maschinellen Verarbeitbarkeit". Die Explizität der Spezifikation fasst die Konzepte, Ideen, Eigenschaften und verbindenden Relationen zusammen. Weiters wird auch angemerkt, dass das gemeinsame Verständnis implizit auf dem Expertenwissen der Kommune basiert. Konzeptualisierung hingegen bezieht sich auf die Tatsache, dass die Zieldomäne gänzlich eingeschlossen wird und bei Bedarf durch konsequente Verwendung der definierten Konzepte erweitert werden kann.

Ontologien werden gemeinhin mittels spezifischer Sprachen abgebildet, den sogenannten Domänen Spezifischen Sprachen, kurz DSL. Diese spezialisierten Sprachmittel fokussieren sich auf die subtilen Unterschiede der diversen Wissens-

domänen. Sie formulieren Regeln um die Wissensverarbeitung in eindeutiger Weise zu ermöglichen. Diese Sprachformen differieren in ihrer Anwendung und Aussagekraft. Abbildung 6.2 veranschaulicht das kanonische Schichtenmodell der *semantic Websprachen*. Ontologiesprachen können prinzipiell in drei Sprachkategorien gruppiert werden:

— Traditionell, zum Beispiel Flogic und Ontolingua

— Web-basiert, zum Beispiel RDF(S) und OWL [CO2000]

— Regelbasiert, zum Beispiel RuleML, SWRL [MM2009]

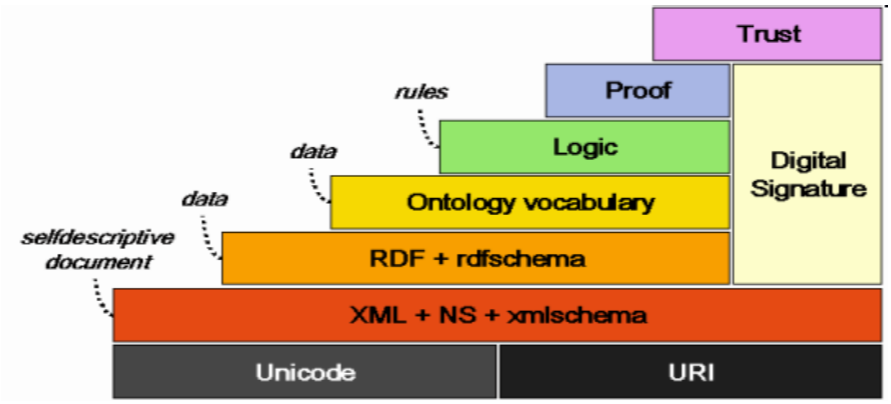


Abbildung 6.2 – W3C Sprachschichten des Semantic Webs

Die Hauptanforderungen an Ontologiesprachen sind, eine wohldefinierte Domänensyntax, Semantik, effizienter Schlussfolgerungsmechanismus (Reasoning) und eine ausdrucksstarke prägnante Idiomatik. Eine weit detaillierte Abhandlung findet sich dazu in [SS2009]. Im Folgenden werden einige der gebräuchlichsten Ontologien der Veranschaulichung wegen kurz dargestellt.

6.2.1.a Simple Knowledge Organization System (SKOS)

SKOS wird derzeit im W3C Framework weiterentwickelt und stellt eine Untermenge von OWL dar. Sein Hauptziel ist die einfache Erarbeitung kontrolliert strukturierter Vokabularien, sowie Navigationsstrukturen für Web 2.0 (Semantic Web). Konzepte können als Ideen und Notationen betrachtet werden. Manche Elemente wie die Fähigkeit Synonyme zu beschreiben, Relationsäquivalenz auszudrücken, Assoziationen und Zusatzinformationen zu beschreiben sind diesem System intrinsisch.

6.2.1.b Metadata Authority Description Schema (MADS)

Das MADS/RDF ist als Datenmodell primär für die Informationswissenschaft entworfen worden. Es spricht somit Einrichtungen wie Museen, Archive und kulturelle Institutionen an. MADS/RDF stellt ein geeignetes Abstraktionsmittel zur Verfügung, um Datensätze im „Machine Readable Cataloging (MARC)“ Format als RDF Tripel für weitere semantische Applikationen wie Triple Store Datenbanken aufzubereiten.

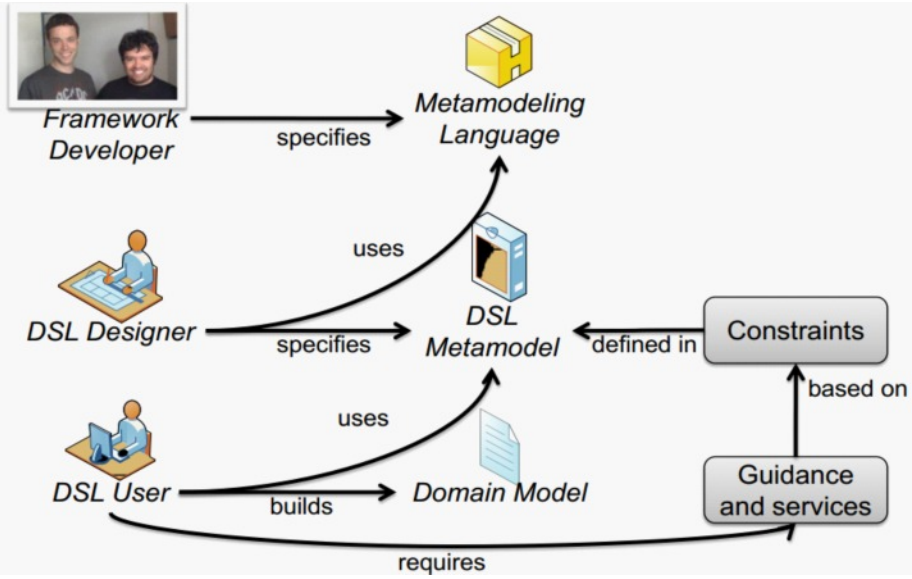
6.2.1.c Dublin Core (DC)

Das Dublin Core Metadaten Vokabular besteht aus 15 Kernelementen die eine große Palette an Ressourcen beschreiben können. Ihr Hauteinsatzgebiet gilt dem begrifflichen Abdecken der Vokabularien von Autoren und Verleger zugleich. Die komplette Menge des Vokabulars, beinhaltet ebenfalls Ressource-Klassen sowie Syntax Schemen zur Kodierung (syntax encoding). Diese Ontologie erfährt in letzter Zeit starken Zuspruch, und ist mittlerweile auch stark erweitert worden. Ab Version 1.1 stellen die 15 Kernelemente eine Untermenge der neuen DCMI[1] Terme dar.

6.2.2 Wann sollte eine eigenständige DSL entwickelt werden ?

Um eine Domäne zu modellieren kann man sich folglich, entweder einer bestehenden Ontologie bedienen die, abstrakt gesehen, nichts anderes ist als eine konventionelle Vokabeldefinition einer vereinbarten Domäne, oder aber auch eine „Neudefinition“ eines abgegrenzten Wissensgebietes selbst entwickeln.

Die einfachste Antwort liefert [MM2005] – Eine DSL sollte dann entwickelt werden, wenn immer es notwendig erscheint ein Problem zu lösen, das einer Problemfamilie entstammt wo erwartet werden kann, dass in naher Zukunft vermehrt härtere Probleme dieser Klasse in Erscheinung treten, die es zu lösen gilt. Die Vorbedingungen um eine Ontologie schlussendlich auf eine DSL zu transformieren, wie es zum Beispiel das *Ontology2DSL* Framework verlangt, ist ein solides ganzheitliches Verständnis der Zielontologie. Der Sprachdesigner sollte auch die Historie dazu kennen, warum und wieso sie entworfen wurde, und vor allem auch natürlich ihren gedachten Einsatzzweck. In den meisten Fällen jedoch, überlappen sich die Anforderungen bezüglich einer DSL und einer Ontologie nicht in allen Punkten. Es kann somit eine einzige Ontologie, beispielsweise *HRO*, verwendet werden, um mehrere DSL's zu entwickeln, vergleiche dazu [ČI2000]. Abbildung 6.3 stellt Ontology2DSL vor.



6.3 – Abbildung einer Ontologie auf eine DSL

[1] Dublin Core Metadata Initiative

Die Vorteile der Benutzung einer bestehenden Ontologie sind offensichtlich, so sind Argumente der Standardisierung, einer stetigen Weiterentwicklung und ontologischen Ausbaues einer *Community*[1], sowie bereits bestehender Toolchains und Konvertierungswerkzeugen definitiv überzeugend.

Jedoch ergeben sich auch Nachteile darin, wie der eingeschränkte „Individualismus“ bei bereits bestehenden Frameworks oder aber auch im Falle einer zu „jungen Domäne“, sprich wo die definierten Konzepte den Ontologieraum noch zu wenig erweitert haben und deswegen noch zu restriktiv sind. Teilweise sind auch komplexe semantische Übersetzungen von Nöten um aus einem generischen Ansatz, speziell bei Wissensmanagement Systemen, ausbrechen zu können.

Bei einer selbst erstellten Domäne ergibt sich als Vorteil der Freiheitsgrad des Entwurfs. Die eindeutige Abbildung eines Vokabulars und Erweiterbarkeit dessen, eine benutzerfreundliche Repräsentation und Schnittstelle der originär entstandenen Werkzeuge sowie die nachträgliche potenzielle Überführung der ausgereiften Domäne auf eine bereits bestehende Ontologie, ein *Ontologiemapping* also. Es sollen auch die Nachteile nicht verschwiegen werden, wie ein deutlicher Mehraufwand in der Erstellung, keine *per se* Standardisierung und meist Neuerstellung sämtlicher Werkzeuge der benötigten Toolchain[2].

Der Vorteil domänenspezifischer Sprachen um „auf natürliche Weise“ Abfragen zu deklarieren, sprich nur das Was und nicht das Wie zu spezifizieren, ist im Vergleich zu General Purpose Languages (GPL) signifikant [FM2010].

Erstens, der Parser anhand der Eingabespezifikation – es ist für ein menschliches Individuum intuitiver als eine komplexe technisch verwobene Syntax zu verwenden. Zweitens kann eine extern generierte DSL bei Falscheingaben klare domänenspezifische Fehlerausgaben erzeugen. Drittens, auch aufgrund der Prägnanz, dient die Eingabe, der “Satz”, auch als eine Art der Dokumentation zu gegebenen Rohdaten. Letztlich aufgrund des deklarativen Charakters können auf einfachere Art und Weise nicht nur der Parser, sondern auch Validierungstools, erweiterte Ausgaben, Statistiken, Transformatoren und vieles mehr generiert werden. In anderen Worten, es kann daraus eine „Toolchain“ an nützlichen Werkzeugen entstehen.

6.3 Das RBMSS Vokabular

Es kann somit festgehalten werden, dass eine Ontologie von RBMSS aufgrund der Heterogenität der Expertengruppen und des oft *nicht* gemeinsamen Verständnisses einer Wissensdomäne stark von der klassischen Ontologiekonzeptualisierung abweicht. Ein *gemeinsamer Nenner* ist sehr wohl gegeben, beschränkt sich jedoch sehr auf wenige Artefakte. Um auch den Anforderungen der Anwender entgegenzukommen muss die Frage der Erfassungs- und Einarbeitungszeit im Detail betrachtet werden. Eine allzu steile Lernkurve erweist sich im Alltagsgeschäft eher

[1] Eine Gemeinschaft von Entwicklern die im Einklang ein Projektziel, zumeist in unentgeltlicher Form als sogenannte „open source Projekte“ verfolgen.

[2] Systematische Ansammlung benötigter Software-Werkzeuge

als *störend*, wobei das System den Anspruch auf Unterstützung hinsichtlich der Geschäftspraktiken des IT-Projektmanagements erhebt.

Fazit, wie schon in der Einleitung erwähnt und nach einer intensiven Diskussion mit allen Beteiligten des Projektes RBMSS wurde mehrheitlich beschlossen, nicht eine allumfassende Konzeptualisierung durchzuführen um das gesamte Wissensgebiet im Vorfeld abzudecken, sondern vielmehr ein stetige wachsende Menge an *Primitiven* parat zu haben, die jedoch eine hohe Flexibilität besitzen und gegebenenfalls *einfach und konsistent* erweiterbar sind. Dies distanzierte die Überlegungen von komplexen Ontologie und Sprachabhängigkeiten und führte hin zu Schlüsselwörter die über Argumente parametrierbar sind. Ein neuer Ausdruck hat sich in diesem Zusammenhang geprägt, die sogenannte *Schlüsselwortpragmatiken*.

Die Vorteile liegen hier vor allem in den Aspekten der

— einfachen Erlernbarkeit bedingt durch die Konsistenz und Klarheit der atomaren Schlüsselwortsignatur

— intuitiven Anwendbarkeit, durch gewohnte algebraisch ähnliche Ausdrucksformen

— offen in ihrer Erweiterbarkeit hinsichtlich neudefinierter Schlüsselworte ohne das bestehende System in eine Inkonsistenz zu führen

— standardisierte Erfassung und Extraktion durch reguläre Ausdrücke ohne Behandlung einer expliziten Grammatik

Der Schlüsselwortpragmatismus besitzt folgende Grundform:

Namensraum.Schlüsselwort(Spezifikation)

Das Wort *Namensraum,* getrennt durch einen Punkt stellt eine projektbezogenen Unterteilung dar, wie zum Beispiel ein Subprojekt namens GUI. Das *Schlüsselwort* hingegen ist eine der kollaborativ vereinbarten Schlüsselwörter, sie repräsentieren somit das gemeinsame Verständnis so wie es den Grundsätzen der Ontologie entspricht. Die *Spezifikation* ist ebenfalls abhängig vom Schlüsselwort, ein Parameter der den Sachverhalt näher spezifiziert. Er kann vor allem auch im direkten Zusammenhang mit regulären Ausdrücken, genutzt werden oder auch Mischformen daraus. Diese Art der Syntax ist gewohnt, einprägsam und besitzt auch *homo-ikonische Züge*[1].

Exkurs „Homo-Ikonik“

Unter diesem Ausdruck wird die Selbstabbildbarkeit verstanden. Es ist eine Eigenschaft der Sprachen, wo die Struktur der eingegebenen Daten gleichzeitig Konstrukte des Programmes abbilden, sprich als solche direkt übernommen werden können. Es ist einfach mit solchen Mustern zu arbeiten, da die Integrität der Semantik in das Programm fast gänzlich unbehandelt übernommen werden kann. Somit ist auch eine optionale Erweiterung jener Strukturen im gleichen Stil eine logische Fortführung des Programmkontexts.

[1] Siehe dazu http://de.wikipedia.org/wiki/Homoikonizit%C3%A4t

Zurück zukommen auf das ursprüngliche Thema, die Nutzung einer bestehenden Ontologie ist eine Notwendigkeit, die andere ist die Verfügbarkeit einer passenden Toolchain von der Notation bis hin zur Darstellung.

Auch hier wird ebenfalls der pragmatische Weg beschritten. Das *Mapping der Ontologie* auf das bestehende Versionskontrollsystem erfolgt, wie in Abbildung 6.4 schematisch, ohne weitere Angaben detaillieren zu wollen dargestellt. Die Benutzung erfolgt durch einen bereits bestehenden, wie auch stark von der Community vorangetriebenen, graphischen Aufsatz, namens *TortoiseHg*[1]. Diese Vorgehensweise bedient sich sozusagen in eklektischer Manier einem Wirtsystem mit dem Hintergrund einer stetigen getriebenen Weiterentwicklung und gleichsam standardisierten unternehmenübergreifenden „Look & Feel“ der Eingabeschnittstelle.

Die Visualisierung der Artefaktausgaben erfolgt mittels standardisierter Techniken wie eine SVG[2] Darstellung über gängige Browsertechnologien oder auch im Bedarfsfall via explizite Werkzeuge. Als Zwischenformat steht stets die korrespondierende RDF/XML[3] Serialisierung zur weiteren potenziellen Maschinenverarbeitung zur Verfügung.

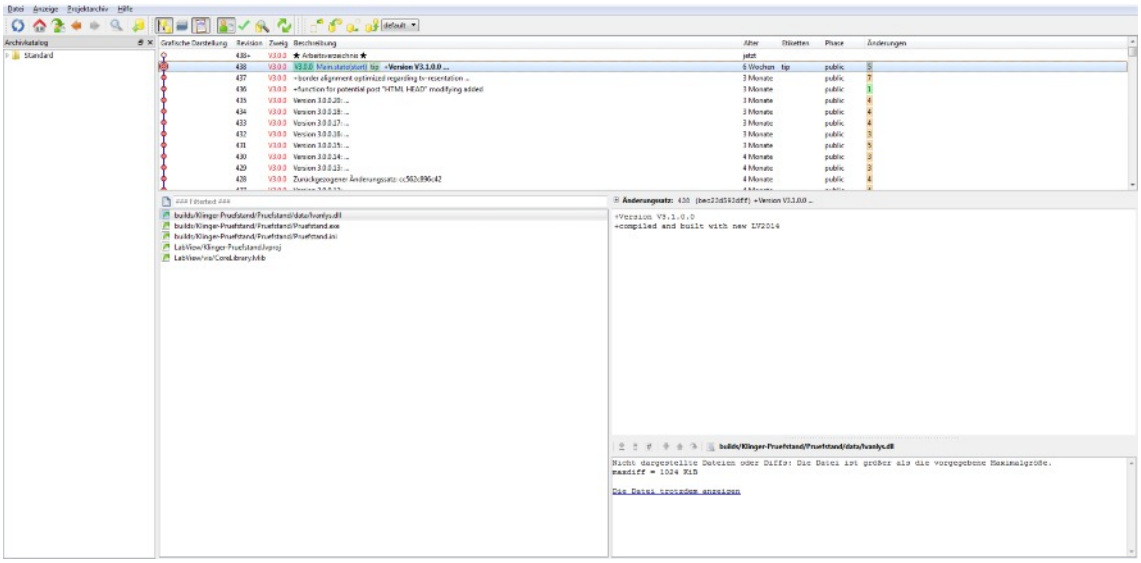


Abbildung 6.4 – graphische Endbenutzerschnittstelle TortoiseHg

6.3.1 RDF als Metadatenauszeichnungsprache

Den Kern der visuellen Repräsentation stellt das generierte RDF/XML Dokument dar. Das Datenmodell [KG2004] ist ausreichend generisch gehalten um die verschiedensten Elemente einer *losen* Begrifflichkeit zu repräsentieren. Viele solcher Entitätsbeschreibungen widerspiegeln sich in Web-Applikationen. Eine Vielzahl an Tools und Bibliotheken stellt sich dem Benutzer, und der Bedarf an sogenannten triple-stores wie am Beispiel des *OpenLink Virtuoso Server* [EO2007]

[1] http://tortoisehg.bitbucket.org/

[2] SVG: Scalable Vector Graphics

[3] RDF/XML: Resource Description Framework in XML-Form serialisiert

zu sehen ist, sind stark im Zunehmen [AD2007]. Gerade deswegen erscheint RDF als geeignete Abstraktionssprache zur Abbildung von Metadaten.

RDF stellt keine harten Anforderungen an relationale Typen, im Gegensatz zu den klassischen Ontologiesprachen. Dieser Freiheitsgrad fördert zum Beispiel das publizieren im WWW[1], das Austauschformat bedingt trotz allem auch ein gemeinsames Vokabular wenn verschiedene Applikationen über die gleiche Datenbasis verfügen. Das RDF-Modell kann in diesem Zusammenhang als ein Strom willkürlich vereinbarter Relationen auf Ressourcen verstanden werden.

Es wird die gesamte Information in Form von Tripeln dargestellt. Das Subjekt, das Prädikat und das Objekt begründen die atomare Information die dieser Aussage innewohnt. Diese Aussagen, auch RDF-Statements genannt, sind ihrer Anzahl nach unbegrenzt und sofern das Vokabular nicht vereinbart wurde auch in ihrem semantischen Kontext unbeschränkt. Die Grundidee in diesem Entwurf liegt in der Beschreibung der Ressourcen als Anlehnung an natürliche Sprachen. Das Subjekt identifiziert die Ressource, das Prädikat hingegen die zu beschreibende Eigenschaft, die Objektcharakteristik schlussendlich den tatsächlichen Inhalt.

Ein RDF-Graph drückt somit ein Faktum aus, ein Wissensfragment das durch den Autor behauptet wird. Dieses Zusammenspiel wird auch durch die *Open World Assumption* (OWA) ausgedrückt, die grundsätzlich jeder Aussage Gültigkeit schenkt, ansonsten müsste sie bewiesenermaßen falsch sein. RDF macht auch keine Annahmen über die Ressourcen, es beschränkt sich lediglich auf eine IRI-Referenzierung.

Das Framework besteht aus einer minimalen Menge an Sprachkonstrukten die wiederum indentierter semantischer Natur sind. Die Kontrollkonstrukte beschränken sich im Wesentlichen auf geordnete und ungeordneten Gruppierungen in Form von Listen, Sequenzen, Säcken (Bags) und so weiter. Durch diese einfache Ausdrucksform des Triples gibt es eine Vielzahl gängiger Serialisierungsformate. Allen voran steht sicherlich das XML-Format, welche die am häufigste verwendete Art darstellt. Es gibt jedoch auch für den Menschen intuitivere Formen, wie Turtle, N-Triples oder auch in der gängigen JSON-Notation.

Eines fehlt ihnen allesamt eine Ausdrucksform für die Kardinalität der beschriebenen Menge, einer der Hauptkritikpunkte die OWL unter anderem adressiert. Diese schwache Ausdrucksform ist wahrscheinlich einer der Gründe warum diese Konzepte in RDF direkt so selten verwendet werden, da man häufig über Anwendung der „Blank Node“ Technik in konsistenter Weise vielleicht mit etwas weniger Ausdrucksstärke, aber ebenfalls gleicher Semantik, erreicht.

Eine weitere Ausdrucksform die jedoch nur selten genutzt wird, ist jene der Reifikation. Darunter versteht man eine *Aussage über eine Aussage* zu formulieren. Ein Beispiel dafür wäre „*Ressource B behauptet Ressource A hätte die Relation R mit dem Objekt O.*“ Diese Konstrukte werden aufgrund ihrer speziellen Behandlung in triple-store Datenbanken die der Persistierung von RDF Aussagen dienen, nur

[1] WWW: Acronym für World Wide Web

spärlich verwendet. Siehe dazu auch die plakative Darstellung namens „the big ugly“ [PS2003a].

Ein weiterer interessanter Aspekt ist, dass das RDF Schema (RDFS), also die Validierungssprache von RDF, in einer rekurrenten Sprachbeziehung zu RDF steht. Sie verwendet RDF im Rückgriff dazu um das Vokabular zu erweitern. Damit werden Klassen (class) definiert und deren Eigenschaften (property), die in einer RDF Description mit Werten belegt werden können. Auch die Ausdrucksstärke von RDFS ist jedoch stark begrenzt, so gibt es nur Domain- und Range Einschränkungen von Classes und Properties und keine einfachen Axiome wie Reflexivität, Symmetrie und Transitivität.

Abschließend ein abstrakt formuliertes Zitat über die RDF-Ressource:

“The things denoted are called ‘resources’, but no assumptions are made here about the nature of resources; ‘resource’ is treated here as synonymous with ‘entity’, i.e., as a generic term for anything in the universe of discourse.” [Hay04]

6.4 Typisierung des Namensraumes von RBMSS

Um nun ein geeignetes Vokabular für die Ontologie von RBMSS zu entwickeln, nutzen wir, wohlbekannte Ausdrücke bestehender Ontologien und ergänzen es gegebenenfalls durch eigens definierte Formen. Da diese neue Typenvereinbarung jedoch in Bezug auf die Ressourcen gesondert referenzierbar sein muss, wird ein neuer Namensraum erschlossen.

Der implizite Metadatenbestand von Versionskontrollsystemen stellt bereits einen reichen Fundus dar. So sind beispielsweise Informationen wie der Name des Autors und gegebenenfalls auch Kontaktadressen bereits fixe Bestandteile der Datensätze. Der Zeitstempel des *eincheckens*[1], also jener Vorgang wo bestimmte Softwareänderungen in kommentierter und versionierter Form in das Repository eingepflegt werden, die dadurch auch jederzeit wieder rekonstruierbar sind, ist ebenfalls eine zentrales Artefakt der Metadaten.

Es ist damit intuitiv nachvollziehbar, dass jene Art des Datum[2] keine neue Ausdrucksbildung benötigt, sondern sofern möglich, auf bestehenden Ontologien aufsetzt damit einerseits der Namensraum nicht durch semantische Doppelbelegungen unnötigerweise *verschmutzt*[3] wird, und andererseits werden bereits häufig benutzte Ontologien in der Praxis besser verstanden und umgesetzt. Eine oft verwendete Terminologie ist der Dublin Core, kurz DC, eine Begriffswelt die ihre Ursprünge im Bibliothekswesen hat und bereits im Kapitel 6.2.1.c kurz dargestellt wurde.

---

[1] Ein synonymes Wort für commit

[2] In diesem Kontext als Singular von Daten zu verstehen

[3] Namespace pollution

RBMSS benutzt prinzipiell eine Untermenge der 15 Kernelemente wie sie in der Version 1.1 der DCMI[1] beschrieben sind, verwendet jedoch auch wenn notwendig die Erweiterungen der DCMI „Metadata Terms“ die zur Verfeinerung der Beschreibungen dienen, da ansonsten die Ausdruckstärke zu gering wäre. Ein Vorteil vom Dublin Core ist das ausdrückliche Angebot hinsichtlich Verlinkung zu anderen Ontologien um die Semantik besser abbilden zu können [PS2003b].

So wird bei DCMI in dem Passus *„Dublin Core and Linked Data“* explizit in folgendem Zitat darauf hingewiesen:

„Though the statements we make about the person are not using Dublin Core properties they are non the less DC compatible, because in the namespace these terms are explicitly declared to be properties - one of the four metadata terms Dublin Core knows - and have been assigned a proper URI (see Dublin Core URIs and namespaces). This means we may use other properties than Dublin Core Terms as long as these terms are Dublin Core compatible.

This way Dublin Core provides the grammar for a "metadata pidgin for digital tourists", a grammar that allows to merge terms from different vocabularies of different communities in one language and may be used to display both simple and complex issues. With this grammar Dublin Core provides a mechanism for extending the Dublin Core term set for additional resource discovery needs expecting that other communities will create and administer additional metadata sets, specialized for the need of their community.“[2]

Insgesamt bedient sich das System konkret 8 Elementen des DC und definiert weitere 4 Elemente im eigenen Namensraum RBMSS. Da DCMI-Terme bezüglich detaillierter Kontaktdaten keine eindeutigen Verfeinerungen anbieten und gerade das VCARD-Vokabular mittlerweile integrierter Bestandteil von Kommunikationssoftware, wie Mail- und Kalenderapplikationen, geworden ist, verwendet das System bei der Angabe des Personennamens sowie seiner Kontaktmöglichkeit die ausdrucksstärkere Variante VCARD.

Die folgende Darstellung in Tabelle 6.1, stellt die insgesamt 14 verwendeten Metadatenartefakte und deren Bedeutung in Zusammenhang mit den Ontologievokabularien der verschiedensten Domänen dar.

[1] Dublin Core Meta Initiative http://dublincore.org/documents/dces/

[2] Dublin Core User Guide http://wiki.dublincore.org/index.php/User_Guide

Kante	*Zielobjekt*	*Semantik*	*Namensraum*
dcterms:type	RBMSS-Knoten	Eigenschaft ausgehend vom Revisionsknoten des Repository	DC-Core
dcterms:date	Commit- und rbmss:state Datum	Datum des commits und der Teilprojektstati	DC-Core
dcterms:isPartOf	Literalobjekt	Teil von Zweig und oder Aufgabe	DCMI-Terms
dcterms:text	Literalobjekt	Text des Schlüsselwortarguments	DCMI-Terms
dcterms:references	URI-Knoten	Revisionsknoten	DCMI-Terms
dcterms: isReferencedBy	Literalobjekt	Text der Repository Tags	DCMI-Terms
dcterms: sizeOrDuration	Numeral	komplette Intervalldauer	DCMI-Terms
dcterms:extent	Numeral	Kostenmetrik	DCMI-Terms
vcard:name	Literalobjekt	Name des Autors	VCARD
vcard:email	Literalobjekt	Email-Adresse	VCARD
rbmss:state	Datumsobjekt	Zustand des Teilprojektes	RBMSS
rbmss:enquiry	Literalobjekt	unidirektionale Anfrage	RBMSS
rbmss:dependency	Literalobjekt	Abhängigkeitspfad	RBMSS
rbmss:knowledgeSource	Literalobjekt	Wissensressource	RBMSS

Tabelle 6.1 – verwendete RDF Eigenschaften und deren Namensräume

Die klassische Namensraum-Referenzierung über den URI-Mechanismus wird für die Einbindung der verschiedensten Vokabularien konsequent angewandt. Über das W3C RDF-Validator Service[1] kann das Dokument auch hinsichtlich seiner Konsistenz validiert werden wobei dies eigentlich redundant ist, da das verwendete RDF-Modul an und für sich keine illegale Syntaxanwendung zulässt.

Damit der neu generierte Namensraum RBMSS von den diversen Anwendungen auch eindeutig referenziert werden kann, wurde der URI ebenfalls unter einem Service zur persistenten URL Abbildung, registriert. Das Service PURL[2], welches auch von DCMI genutzt wird, dient genau dem Zweck und bietet zur weiteren Erkundung des adressierten Vokabulars den HTTP-Response Status Code 303 an. Wie in RFC 7231 vereinbart, ist die Intention folgende:

"A 303 response to a GET request indicates that the origin server does not have a representation of the target resource that can be transferred by the server over HTTP. However, the Location field value refers to a resource that is descriptive of the target resource, such that making a retrieval request on that other resource might result in a representation that is useful to recipients without implying that it represents the original target resource."[IETF2014]

Somit kann eine Anwendung automatisch darauf reagieren und bei Bedarf weitere Informationen daraus beziehen. Wie in Abbildung 6.5 ersichtlich, nutzt RBMSS zur Erschließung der verschiedenen Vokabularien folgende Namensräume der RDF/XML Serialisationsform.

```
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:dcterms="http://purl.org/dc/terms/#"

xmlns:rbmss="http://purl.org/rbmss/terms/1.0/#"

xmlns:vcard="http://purl.org/vcard/elements/1.1/#"
```

Abbildung 6.5 – genutzte Namensräume der RDF/XML Serialisation

Wie in obiger Darstellung unschwer zu erkennen ist, lehnt sich die Referenzierungsart des RBMSS Namensraumes stark an die gängigen Konventionen an. Wobei das Hash-Zeichen, das Vokabular über die konventionelle *HTML-Anker* Technik adressiert. Implizit wird auch durch die optionale Konvertierung zu SVG ein weiteres Vokabular verwendet. Diese Art der Referenzierung, dient ebenfalls einer Applikationsklasse, nämlich den Browsern. Sie verwenden hingegen zur Diskriminierung der Vokabeln eine Dokument Type Definition, kurz DTD, die einen Namensraum zwecks Validierung benötigt.

Im Fall von RBMSS wird hier zusätzlich die *SVG-V1.1 Full Syntax* benutzt. Diese Vereinbarung wird durch den Ausdruck *<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">* am Dokumentenanfang eingefügt.

[1] http://www.w3.org/RDF/Validator/

[2] https://purl.org/docs/index.html

Das selbst erstellte RBMSS-Vokabular, deklariert sich aus dem RDF Schema der Klasse *class* die wiederum eine Ableitung der generischen *Ressource* darstellt. Dies ist auch eine notwendige Bedingung. Die Entitäten von RBMSS entsprechen extrahierten Ressourcen des Repositories, und das genutzte DC Vokabular den Eigenschaften *(Properties)* beziehungsweise den Relationen zueinander.

Die folgende Abbildung 6.6 deklariert das konkrete RBMSS-Vokabular mittels *RDFS* und dem *DCTERMS* Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY rdfns 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
   <!ENTITY rdfsns 'http://www.w3.org/2000/01/rdf-schema#'>
   <!ENTITY dcns 'http://purl.org/dc/elements/1.1/'>
   <!ENTITY dctermsns 'http://purl.org/dc/terms/'>
]>

<rdf:RDF xmlns:dcterms="http://purl.org/dc/terms/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<!--rbmss:state-->
<rdf:Description rdf:about="http://purl.org/rbmss/terms/state">
<rdfs:label xml:lang="en">State</rdfs:label>
<rdfs:comment xml:lang="en">A group of Repository States.</rdfs:comment>
<dcterms:description xml:lang="en">Examples of State Class include groups seen as start stop timeintervals in terms of project progress.</dcterms:description>
<rdfs:isDefinedBy rdf:resource="http://purl.org/rbmss/terms/1.0/#state"/>
<dcterms:issued rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2014-01-16</dcterms:issued>
<dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2014-07-07</dcterms:modified>
<rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<dcterms:hasVersion rdf:resource="http://http://presulli.at.vu/usage/terms/history/#State-003"/>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<!--rbmss:dependency-->
<rdf:Description rdf:about="http://purl.org/rbmss/terms/dependency">
<rdfs:label xml:lang="en">Dependency</rdfs:label>
<rdfs:comment xml:lang="en">A group of Dependencies.</rdfs:comment>
<dcterms:description xml:lang="en">Examples of Dependency Class include groups seen as dependencies with other related tasks.</dcterms:description>
<rdfs:isDefinedBy rdf:resource="http://purl.org/rbmss/terms/1.0/#dependency"/>
<dcterms:issued rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2014-01-22</dcterms:issued>
<dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2014-07-07</dcterms:modified>
<rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<dcterms:hasVersion rdf:resource="http://dublincore.org/usage/terms/history/#Dependency-002"/>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<!--rbmss:enquiry-->
<rdf:Description rdf:about="http://purl.org/rbmss/terms/enquiry">
<rdfs:label xml:lang="en">Enquiry</rdfs:label>
<rdfs:comment xml:lang="en">A group of Enquiries.</rdfs:comment>
<dcterms:description xml:lang="en">Examples of Enquiry Class include groups seen as enquiry references similiar to an upper layer solicite response.</dcterms:description>
<rdfs:isDefinedBy rdf:resource="http://purl.org/rbmss/terms/1.0/#enquiry"/>
<dcterms:issued rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2014-01-22</dcterms:issued>
<dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2014-05-23</dcterms:modified>
<rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<dcterms:hasVersion rdf:resource="http://http://presulli.at.vu/usage/terms/history/#Enquiry-002"/>
<rdfs:domain rdf:resource="http://purl.org/dc/terms/MethodOfAccrual"/>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<!--rbmss:knowledgeSource-->
<rdf:Description rdf:about="http://purl.org/rbmss/terms/knowledgeSource">
<rdfs:label xml:lang="en">KnowledgeSource</rdfs:label>
<rdfs:comment xml:lang="en">A group of KnowledgeSources.</rdfs:comment>
<dcterms:description xml:lang="en">Examples of KnowledgeSource Class include groups seen as references to project related knowledge sources.</dcterms:description>
<rdfs:isDefinedBy rdf:resource="http://purl.org/rbmss/terms/1.0/#knowledgeSource"/>
<dcterms:issued rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2014-01-22</dcterms:issued>
<dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2014-02-20</dcterms:modified>
<rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<dcterms:hasVersion rdf:resource="http://http://presulli.at.vu/usage/terms/history/#KnowledgeSource-001"/>
<rdfs:domain rdf:resource="http://purl.org/dc/terms/MethodOfInstruction"/>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>
</rdf:RDF>
```

Abbildung 6.6 – RDF-Schema der RBMSS Schlüsselworte

Dem aufmerksamen Leser wird auffallen, dass der nachfolgende anschauliche SVG-Graph durch obiges Schema mittels *RBMST* übersetzt wurde. Das Werkzeug wurde damit zur Selbstdokumentation der eigenen vereinbarten RDF-Klassen herangezogen. Ein weiterer Vorteil der sich dadurch ergibt, ist dass die obige Notation auch implizit einem Validierungsprozess unterzogen wurde, denn ansonsten würde bei grober syntaktischer Fehlanwendung die Darstellung nicht erzeugbar sein und der Prozess mit einer Fehlermeldung terminieren. Die folgende Abbildung 6.7 veranschaulicht das RBMSS-Schema in Form eines SVG-Graphen.

Abbildung 6.7 – RBMST generierter SVG-Graph des RBMSS-Schemas

Teil III
Die Konzeptionen und ihre Implementierung

7 Kapitel

Die Basisentscheidungen der Implementierung

"Es ergibt keinen Sinn, präzise zu sein,
wenn man überhaupt nicht weiß, wovon man spricht."
John von Neumann

Schwerpunkte in diesem Kapitel :

- Das Projekthandbuch als Anforderungsbasis
- Die Wahl des Versionskontrollsystems
- Die Basismechanismen des Wirtssystems
- Sinnvolle Konventionen von RBMST
- Data Mining und Reasoningmechanismus

7.1 Das Projekthandbuch als Anforderungsbasis

Dieses essentielle Hilfsmittel des Projektmanagements soll als Basis zur Betrachtung der Einsatzmöglichkeit von RBMST dienen. RBMST kann als eine Instanz von RBMSS aufgefasst werden. Der Haupteinsatzzweck ist die Unterstützung des Projektmanagements in seinen zyklischen Projektabgleichsitzungen, wie zum Beispiel bei Projektstatusmeetings und Festlegungen von *Roadmaps*[1]. Der Zweck und Aufbau eines Projekthandbuches [PM2011] widerspiegelt teilweise die Anforderungen an das System. Es dokumentiert in kompakter Weise den

— groben Projektumfang mit geplanten Start und Endterminen

— Projekt Haupt- und Unterziele

— Zuständigkeiten und Abhängigkeiten in den Projektphasen

— Projektstruktur- & Arbeitspaketplan

— Projekteinsatz- & Projektmeilensteinplan

— Projektkoordination & Projektcontrolling

ohne jedoch in den jeweiligen Planungsstufen Detailvorgaben zu machen. Somit stellt dieses Handbuch eine Projektquerschnittsbetrachtung auf abstraktem Niveau dar, mit dem Fokus einer ganzheitlichen Sicht des Projektlebenszyklus. Diese

[1] http://de.wikipedia.org/wiki/Roadmap

logische Sichtweise eröffnet nun die Gelegenheit einige wichtige Projektfaktoren in einer abstrahierten Sichtweise des Projektmanagements so darzustellen, dass zwar die Kernfaktoren zur Geltung kommen können, jedoch den Anspruch der adäquaten Übersichtlichkeit behalten.

7.2 Die Wahl des Versionskontrollsystems

Um diese Referenzimplementierung auch in der Praxis erproben zu können, wurde der Fokus auf ein Versionskontrollsystem gesetzt, welches die folgenden Anforderungen in absteigender Priorität adressiert:

— portable open source Technologie mit starker Verbreitung im Praxisumfeld

— flexible Architektur um individuellen Projektanforderungen zu entsprechen

— einfache Erweiterbarkeit durch eigene Adaptierung und PlugIns

— intuitive, ergonomische und komfortable Benutzerschnittstellen

Es gibt unter diesen Prämissen sicherlich einige Konkurrenten, allen voran wäre *Git* und *Bazaar* zu nennen, jedoch setzt ein signifikanter Anteil der Industriepartnern offensichtlich *Mercurial*[1] ein, so wie es sich auch bezüglich der Partnerauswahl in der Erprobungsphase feststellen lies.

Das Versionskontrollsystem Mercurial, in weiterer Folge auch als *hg* [2] bezeichnet ist, wie schon in Kapitel 2.1.3 kurz dargestellt, eines der gängigsten dezentralen Versionskontrollsysteme unserer Zeit. Es findet starke Verwendung bei kollaborativ geführten „Multiautor Projekten“ [WD2010] wie Mozilla, Google Chrome, OpenJDK und Python, um nur einige zu nennen. Es sind mehrere Faktoren die den Einsatz von Mercurial forcieren [OB2009]. Ursprünglich aus der Historie heraus hat es den Anspruch als ein Ersatzsystem für die Linux-Kernel Versionsverwaltung, da das damalige *BitKeeper* System nicht mehr unentgeltlich zur Verfügung stand. Es entschied sich jedoch Linus Torvalds für eine eigene Implementation namens Git[3] welches die spezifischen Anforderungen der Community bezüglich Linux-Kernel Versionierungssystematik besser abzudecken scheint [LJ2009].

Hg zeigt sicherlich seine Stärken vor allem im Bereich der Portabilität und Benutzerfreundlichkeit, wahrscheinlich hat letzteres seine zunehmende Verbreitung am stärksten begünstigt. Die Benutzerfreundlichkeit widerspiegelt sich vor allem in der sogenannten *Workbench,* das zentrale Cockpit des grafischen Aufsatzes *TortoiseHg*, über welchem man fast alle Aspekte des Systems in übersichtlicher und ergonomischer Weise nutzen kann. Nach positiver Prüfung der prinzipiellen notwendigen Mindestanforderungen bezüglich der Metadaten Injektion, fiel die pragmatische Entscheidung somit auf Mercurial.

---

[1] http://mercurial.selenic.com/

[2] Diese Name leitet sich vom chemischen Element Quecksilber (Hg) ab.

[3] Git is a british slang for a dumb, annoying, or generally unpleasant person.

7.3 Die Basismechanismen des Wirtssystems

Die Mechanismen die im folgenden erläutert werden sind ausschließlich jene, die von RBMST genutzt werden. Sie sind in der Grundfunktionalität von Mercurial vorhanden und bedürfen keiner zusätzlichen 3rd Party PlugIns. Diese Beschränkung auf die Grundfunktionalität ist ein wichtiges Faktum um eine künftige Portabilität von RBMST auf andere Wirtssysteme zu gewährleisten, denn die hier vorgestellten Mechanismen bilden den eigentlichen Grundstock eines jeden soliden Versionskontrollsystems.

7.3.1 Der „seiteneffektfreie“ Injektionsmechanismus

Die erste prototypische Version von RBMST verfolgte den naiven Ansatz der *Beschlagwortung*, also einer Metadaten Injektion, über den normalen Commit-Log Zyklus, einer Injektierung von Schlüsselwörtern in der Log-Nachricht selbst. Diese Vorgehensweise wurde letztendlich in den kollaborativen Sitzungen verworfen, da man die Ansicht vertrat, dass die Zuverlässigkeit der Metadatenbasis in Bezug auf „backouts“ und „rollbacks“ zu vage, und eine im Bedarfsfall angestrebte Beweisführung zu komplex wäre, es stellte sich die Forderung nach seiteneffektfreier Injektion.

Die beiden Bezeichnungen *backout* und *rollback* sind jeweils versionskontrollsystemspezifisch, und werden in anderen VCS teilweise verschieden benannt. Die Semantik ist jedoch stets dieselbe, so bezeichnet ein *rollback*, ähnlich wie in relationalen Datenbankmanagementsystemen (RDBMS), ein Rücksetzen des ausschließlich letzten Vorganges (commits) – es kann als Spezialfall des *backouts* betrachtet werden. Ein *backout* hingegen kann eine *beliebige* Anzahl von Vorgängen rücksetzen, muss jedoch fallweise wieder mit der Letztversion verschmolzen (gemergt*)* werden, da in diesem Fall die Datenänderungen auch nicht strikt linear verlaufen können. Damit sollte offensichtlich werden, dass solche Operationen direkte Manipulationen an der Datenbasis vornehmen, und somit unerwünschte Seiteneffekte an den Produktivdatenbestand hinterlassen können.

Des Weiteren sollte das System einen unidirektionalen Charakter, in Form eines „bottom-up Paradigmas“ besitzen. Dieses entspricht auch dem Induktionsprinzip, also von etwas Konkreterem auf etwas Allgemeineres schließen zu können, ohne den Informationsfluss zu ändern. Anders formuliert, die untere konkrete Ebene, die der Techniker, injektiert ausschließlich jene Metdaten die auf der oberen abstrakteren Ebene, die der Manager, bedarfsorientiert extrahiert werden können.

Dies wurde deswegen von beiden Parteien bejahend angenommen, da einerseits mehrheitlich das Management nicht den Umgang eines neuen Tools erlernen wollte, andererseits die Beteiligten auf der technischen Seite bereits die Bedienung des VCS-Werkzeuges verinnerlicht hatten und auch teilweise im Falle unsachgemäßer Bedienung ein berechtigtes Unbehagen etwaiger Systeminkonsistenzen des Versionsverlaufes befürchteten.

7.3.1.a Der Bookmark-Mechanismus

Diese Technik erfüllt in jeglicher Art und Weise die Anforderung nach der Freiheit von Seiteneffekten. Der Mechanismus kann als eine Art Zeigersemantik, wie sie aus der Welt der imperativen Programmiersprachen bekannt ist, aufgefasst werden. Die sogenannten Bookmarks, auch Lesezeichen genannt, bilden eine zusätzliche Ebene über den versionsgepflegten Artefakten ab. Sie werden dazu in einer unversionierten Datei namens *.hg/bookmarks* gehalten, und sind somit nicht direkter Bestandteil der Artefakte.

Somit ist auch im Fall von „backouts“ und „rollbacks“ keine Inkonsistenz zu befürchten, da das Lesezeichen anschließend *eradiziert* wird, was bei einer Notation via Log des betreffenden Changesets nicht möglich wäre, da eine der Hauptforderungen an Versionskontrollsystemen, die eigentliche Unveränderbarkeit der History an oberster Stelle steht, und somit gravierende Seiteneffekte hätte.

Insgesamt bietet Mercurial fünf verschiedene Brancharten, auch Zweiggestaltung genannt, an. Es sind jedoch nur zwei die am häufigsten verwendet werden, nämlich der benannte und der anonyme Branch, wobei der letztere einen Nebeneffekt aufgrund einer notwendigen Aktualisierung (Updates) darstellt. Der Bookmark-Mechanismus ist durch sein asynchrones Verhältnis zum commit Zyklus auch wesentlich flexibler, was Vor- und jedoch auch Nachteile mit sich zieht.

Der Vorteil der zeitlichen Entkopplung zum *commit* ist offensichtlich, jedoch wird ein „vergessen“ dadurch natürlich ebenfalls möglich. Eine weitere Eigenart dieser Technik ist auch die Trennung der Gültigkeitsbereiche. Es besteht die Möglichkeit Lesezeichen „lokal“ und „global“ zu führen, damit ist unter der Voraussetzung einer korrekten Anwendung ein bewusstes Propagieren der Zeigerinformation in ein übergeordnetes synchronisiertes Repository geschaffen.

7.3.2 Verschmelzungspunkte im azyklischen Versionsgraphen

Eine essentielle Methodik um Entwicklungsstränge teilen zu können ist die Nutzung von Branches und Tags. Die Erstellung von Zweigen können viele Notwendigkeiten hervorrufen, wie zum Beispiel einen separaten Debug branch um geeignete Ausgaben zur Fehlersuche parat zu haben die jedoch in der Produktionsausgabe nicht vorhanden sein sollten, oder auch eine klassische Trennung der Kernfunktionalität und der Module mit Subsystem Charakter.

Ein häufig verwendeter Zweig ist auch der Hotfix & Test-Branch. In diesem werden parallel zu einer Releaseerstellung ein eigens kreierter Zweig für die Qualitätssicherung bereit gestellt, um so auch schon während des Entwicklungsprozesses losgelöst bedingte Tests und Fehlereingrenzungen, wie in Beispiel von Systemintegrationstests oder Ergonomietests, exerzieren zu können.

Wie später noch gezeigt wird, korreliert die Nutzung von Zweigen stark mit in den Projekten verwendeten Vorgehensmodellen. Abbildung 7.1 stellt beispielhaft Verzweigungen und Etiketten (Tags) graphisch dar.

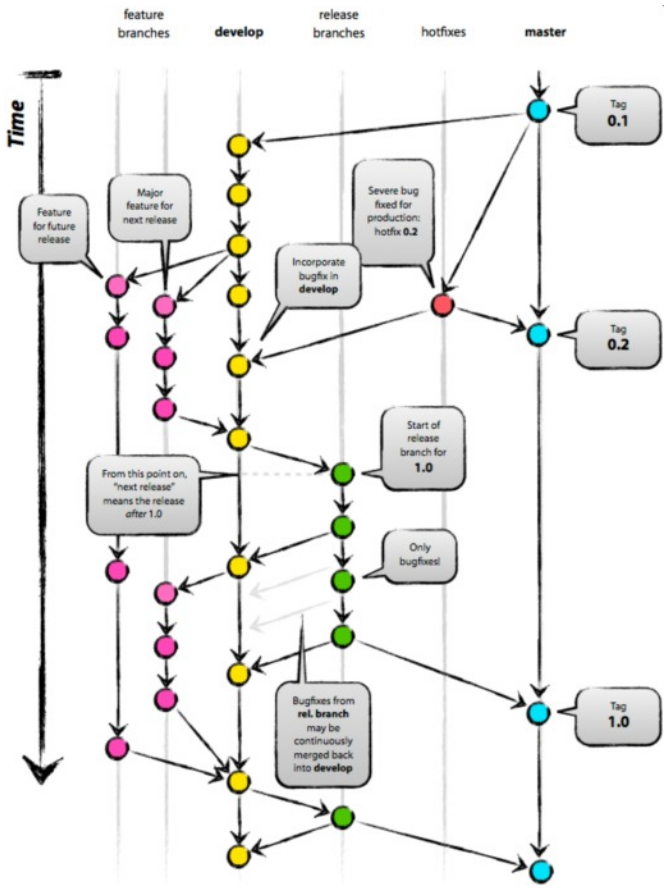


Abbildung 7.1 – Branching & Tagging

7.3.2.a Tagging-Mechanismus

Falls nun die Software die Testanforderungen gemäß der (Teil)Spezifikation erfüllt, wird häufig ein *interner Snapshot* in Form eines sogenannten Tags gemacht, auf den sich nun jeder Projektbeteiligte referenzieren kann. Dieser Mechanismus ist wichtig, da zum Beispiel bei Regressionstests und der Gleichen sonst kein *Anker* als Bezugspunkt vorhanden wäre. Um korrekt diese Technik einzusetzen, bedarf es jedoch auch ein verschmelzen der zuvor entstandenen Branches. Dieser Vorgang des *mergens* stellt somit ein kontrolliertes Vereinen von azyklisch gerichteten Teilgraphen dar, die dann im Erfolgsfall mit einem Tag meist in der Form einer Versionsbezeichnung, zum Beispiel der häufig verwendeten Tripelnotation V1.2.3. etikettiert wird.

Diese Konvention ist sehr gebräuchlich da die erste Zahl die Hauptversion, spricht die vereinbarte Grundfunktionalität auszeichnet, das zweite Element meist als eine Revision eines Submodules verwendet wird und die dritte Version den generellen (bugfix) Patch-Level bezeichnet. Es ist somit mit einer Kombination von Lesezeichen, Zweigen und Tags eine hohe Ausdruckstärke realisierbar, ohne ein externes Werkzeug bemühen zu müssen. Es sei jedoch angemerkt, dass das etikettieren (*taggen)* keinen seiteneffektfreien Effekt darstellt, da das Versions-kontrollsystem ein eigenes Changeset als Datensatzeintrag in der History erzeugt und somit als Artefakt persistent verwaltet.

7.3.3 Die verschiedenen Arten der Projektpartitionierung

Versionskontrollsysteme sind abgesehen von ihrer ureigensten Zielsetzung, der kontrollierten Versionsverwaltung, auch prädestiniert zur logischen Aufteilung eines Software-Projektes. Bedingt durch die Möglichkeiten einer dezentralen Verwaltung besitzt es gegenüber den klassisch zentralisierten Systemen einige Vorzüge. Es bieten sich somit schlagartig Vorteile, wie die Unabhängigkeit von singulären Netzwerkverbindungen, bessere Lastverteilung der Repositories sowie *Sandbox* Entwicklungen, also lokal abgeschottete Entwicklungsversuche die im Erfolgsfall einfach in die eigentliche *Baseline*[1] des Projektes aufgenommen werden, an. Gerade diese Flexibilität verursacht auch eine zusätzliche Komplexität in Bezug auf die Schaffung notwendiger unternehmensinterner Richtlinien zur Nutzung der Infrastruktur.

7.3.3.a Der Workflow

Es ist intuitiv vorstellbar, dass gerade diese Einsatzvielfältigkeit ein großes Portfolio an sinnvollen Arbeitsabläufen (Workflows) zulässt, klassische Arbeitsabläufe finden sich unter [OB2014]. Mehr von Interesse hingegen ist eine projektspezifische Partitionierung in Bezug auf eine deklarative Kennzeichnung durch die angestrebte Kategorisierung. Darunter wird verstanden, dass ein Projekt so aufgeteilt werden kann, dass die Rückverfolgbarkeit der Zuständigkeit in Bezug auf die Teilaufgabe in projektorganisatorischer Manier gewährleistet werden kann. Der Partitionierungsaufwand hängt stark mit der Projektkomplexität, in Bezug auf die Funktionalität (Features), die zeitliche Disponierbarkeit sowie der Teamgrößen ab.

Es besteht somit die Möglichkeit, dass über einen Benutzer- und oder Gruppennamen, in einem Zeitfenster bezogen auf einen Zweig der über die zusätzliche Metatag-Information, *den Task*, feiner zugeordnet werden kann, ein *relativ exakter* Projektabschnittszustand, auch Projektsnapshot, ermittelt werden kann. Diese Partitionierung des Arbeitsablaufes in Kombination mit der Information *getaggter* Versionsstände wäre eine sinnvolle Ergänzung in Projektstatusmeetings beziehungsweise Audits. Vor allem eine akkordierte Vorgehensweise im Einklang mit (Feature)-Roadmaps wäre dazu sicherlich ein projektorganisatorischer Gewinn. Die nachfolgende Abbildung 7.2 veranschaulicht ansatzweise die willkürliche Aufteilung eines Workflows.

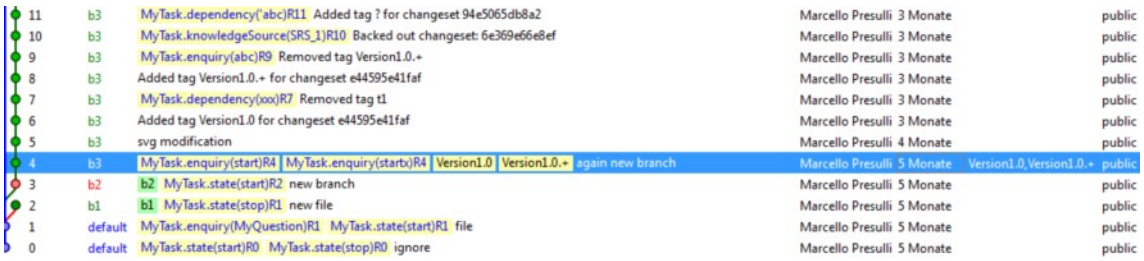


Abbildung 7.2 – eine injektierte Schlüsselwortpragmatik samt Etikett

[1] Oft verwendetes Synonym für den aktuell bestehenden Versionsstand

7.4 Sinnvolle Konventionen von RBMST

Um den projektspezifischen Systemnutzen zu maximieren sollte die Anwenderschaft vor Inbetriebnahme gewisse Konventionen vereinbaren. Diese wirtschaftsinformatischen Vereinbarungen greifen in den verschiedenen Ebenen der Projektorganisation ein. Man kann sinnvollerweise die Konventionen in Hinsicht unterteilen, dabei beansprucht der organisatorische Part sicherlich den Großteil der Vereinbarungen für sich.

7.4.1 technische Konventionen

Die technischen Vereinbarungen sind hauptsächlich lexikalischer Natur und auch notwendig. Denn zumeist jedes System benötigt das Reservieren einiger systemimmanenter Schlüsselwörter, beziehungsweise einiger Symbole aus dem darunterliegenden Alphabet um eindeutige bijektive Abbildungen von Datensätzen gewährleisten zu können. So fällt das ‚:' Symbol das als Separator in der internen Schlüsselnotation zwischen RevisionID:ChangeSetHash dient, sowie die Whitespaces ‚\n' und '\r' die, sofern sie in der Zeile alleinstehend sind, als Ende des Metadatensatzes interpretiert werden, darunter. Vorteilhaft erweist sich, dass bei Eingabe dieser Terminalsymbole die Mercurial Operationen mit einer Fehlermeldung zuverlässig terminieren. Eine potenzielle Inkonsistenz kann damit a priori ausgeschlossen werden.

Um eine Kollision, Mehrdeutigkeit gleichgearteter Metadaten, jedoch in verschiedenen zeitlich versetzen Revisionen zu vermeiden, wird dem Metadatensatz ein hinreichend genügender *Diskriminator* angehängt. Diese *Pseudochecksumme*, die derzeit noch manuell vergeben wird, hat stets die Form *R*xxx, wobei R für Release steht und *xxx* die aktuelle injektierte Revisionsnummer angibt, wie zum Beispiel

Task1.dependency(Task2)***R120***

Das Unternehmen sollte sich dem gemäß im Vorfeld eine Nomenklatur bezüglich ihrer Artefaktbezeichnungen vereinbaren, um nicht während der Betriebsphase unnötigerweise syntaktische Umstellungen durchführen zu müssen.

7.4.2 ökonomische Konventionen

Die ökonomische Betrachtungsweise erschließt den zeitlich optimalen Aufwand in Bezug auf dessen Nutzen. Es gilt somit, ein projektspezifisches *Equilibrium* bezüglich Terminisierung von Projektstatusmeetings in Akkordierung mit den gewünschten Informationsbeständen zu finden. In anderen Worten, sollte vorher geklärt werden, welche Grundinformationen stets verfügbar sein sollten und fallweise welche transienten Informationen ermittelt werden sollten.

Diese Unterscheidung bestimmt auch in einer Art und Weise die mandatorischen Punkte in der Zeitlinie des Projektes wo bestimmte Informationen verfügbar sein *müssen*. Durch diese konventionelle Vorgehensweise, sieht sich auch der

Benutzer des Systems, in der *Bringschuld* einer anstehenden Metadaten Injektion. Dieser Prozess sollte im Stil eines kontinuierlichen Verbesserungsprozesses (KVP) [KC2013] immer effizienter werden und sich letztendlich auf einem *Niveau* stabilisieren.

7.4.3 organisatorische Konventionen

Unter diesen Übereinkünften, die eigentlich auch wirtschaftsinformatisch gekennzeichnet sind, wird der gesamte organisatorische Ablauf unter Betrachtung einer optimalen Systemausnutzung zusammengefasst. Es sollten vorerst, wie schon in Kapitel 7.3.3. diskutiert, die Partitionierung des Arbeitsablaufes stattfinden. Die zuvor als technische Notwendigkeit erörterte Nomenklatur sollte auf ihre syntaktische Gültigkeit in den eingesetzten Argumenten der injizierten Metadaten inspizieren werden.

Das System ist in diesem Bezug, sprich bei Fehleingabe von Argumentdaten *robust,* und verwirft das ungültige Datum fährt jedoch in seiner Exekution fort, da es keine notwendige semantische Verkettung zwischen zwei aufeinanderfolgenden Metadatenargumenten gibt. Dennoch sollte eine etwaige Fehleingabe korrigiert werden, da ansonsten der wahre semantische Inhalt nicht erfasst werden kann. Somit sind auch zyklische Begutachtungen der eingegebenen Metadaten als sinnvoll anzusehen.

Die manuelle Begutachtung ist nur von begrenzter Notwendigkeit, da syntaktische Fehler vom System automatisch gemeldet werden können. Dies ist ein Punkt der in Kapitel 14.2.2.a ausführlich diskutiert werden wird, da er wie einige andere Aspekte auch, mithilfe der Hilfsapplikation namens „*CreateAnnotations*“ bereits während der Eingabe sicher behandelt werden kann.

7.5 Data Mining und Reasoningmechanismus

Zuvor wurden die Interaktionen der verwendeten Kernmechanismen von Mercurial diskutiert. Um nun die Metainformation zu extrahieren muss der zur Verfügung stehende Metadatenbestand in zwei Kategorien unterteilt werden. Die eine Kategorie ist obligatorischen Ursprungs, sie stellt somit Informationen zur Verfügung die pervasiv im System vorhanden sind, die andere Kategorie hingegen ist die Art der *originär* injektierten Metainformationssätze, die fakultativer Natur ist, also nur optional erwartet werden kann. Im Folgenden werden die beiden Kategorien der Reihe nach vorgestellt.

7.5.1 Der obligatorische Metadatenbestand des Wirtsystems

Das Wirtsystem Mercurial, beherbergt intrinsische Informationen in jedem Changeset. Ein *Changeset* ist die logische Gruppierung aller modifizierter Dokumente auf Dateiebene und auch gewisser Pseudoressourcen, wie zum Beispiel das Etikettieren (taggen), während eines *commits*. Es ergeben sich somit zwangsweise Metadaten, die immer denselben Verbunddatentyp besitzen.

```
changeset:   4:e44595e41faf
branch:      b3
bookmark:    MyTask.enquiry(start)R4
bookmark:    MyTask.enquiry(startx)R4
tag:         Version1.0
tag:         Version1.0.+
user:        Marcello Presulli <m.presulli@embed-it.com>
date:        Fri Apr 25 10:24:59 2014 +0200
summary:     again new branch
```

Abbildung 7.3 – der Metadatensatzaufbau von Mercurial

Die obige Abbildung 7.3 stellt die obligatorischen Metadaten eines Mercurial-Changesets in einem Mercurial Schema dar. Wie deutlich zu erkennen ist, setzen sich die Daten durch folgenden Verbundtyp zusammen. Es wird dazu eine C ähnliche Pseudo-Code Notation verwendet:

typedef struct {
String changeset = Integer ‚:' 12 Stellen HexCode (intern 40 Stellen)
String branch = equivalent zu regulärem Ausdruck [_A-Za-z0-9]
String bookmark = gesamter Zeichensatz exklusive ‚:' ‚\r' und ‚\n' verwendbar
String tag = gesamter Zeichensatz exklusive ‚:' , ‚\r' und ‚\n' verwendbar
String user = gesamter Zeichensatz erlaubt
String date = Langformat, kann auf ISO 8601 Formatierung umgestellt werden
String summary = erste Zeile exklusive ‚\r','\n' der gesicherten Log-Information
} Metainfo

Auflistung 7.1 – die logische Verbundstruktur der Metadaten

Auflistung 7.1 stellt die Struktur in wohlgeformten zeilenbasierten Zeichenketten dar. Sie ähneln einer Key : Value Beziehung, wobei dem Schlüssel (Key) eine Liste von Werten zugeordnet werden kann, siehe dazu den Eintrag *tag*.

Weiters ist auch zu bemerken, dass das Benutzerfeld (*user*) erstens eine theoretisch lexikalisch unbeschränkte Zeichenfolge darstellt, und zweitens nicht nur den Namen des Autors festgelegt, sondern bedingt durch seine Freiform, auch die Möglichkeit einer frei formulierten Kontaktadresse zulässt.

7.5.2 Die Metadateninjektion von RBMSS

In den Changesets, die den Schlüssel *bookmark* besitzen, befindet sich ein Wert der von RBMSS vereinbart wurde – diese Einträge stellen somit die zweite Kategorie dar, die originär injektierten Metainformationen. Die Form, die schon in Kapitel 7.4.1 kurz im Zusammenhang mit den technischen Konventionen angeführt wurde, erinnert ein wenig an die Objekt Orientierte Programmierung, kurz OOP.

Das erste Element stellt die Entität (Objekt), das zweite Element getrennt durch einen Punkt ‚.' die angewandte Methode (Receiver) und das dritte Element stellt den eigentlichen semantischen Inhalt (Argument) dar.

Objekt.Methode(Argument)

Abbildung 7.4 – die syntaktische Generalform der Schlüsselwortpragmatik

Wie Abbildung 7.4 suggeriert, ist es vor allem für Softwaretechniker ein intuitives Paradigma, das sofort verstanden werden kann, ohne eine steile Lernkurve[1] meistern zu müssen. Der Fokus der RBMSS basierten Metainjektion ist das gemeinsame Verständnis der sogenannten Schlüsselwortpragmatiken, ein Begriff der sich interessanter Weise im Zuge der kollaborativen Sitzungen als ein Synonym etablierte. Da das System prinzipiell unbeschränkt ist, und sich im Zuge der Projektrealisation der Bedarf nach weiteren *Pragmatiken* ergeben kann, kann es exakt in derselben Weise mit gegebenenfalls erweiterten Schlüsselwortpragmatiken injektiert werden. Das System ist dadurch im *kollaborativen* Kontext des Aufbauwissens stetig erweiterbar.

7.5.3 Der Schlüsselwortpragmatismus

Der Entwurf und die Vereinbarung konsistenter Schlüsselwörter war eine der wichtigsten Zielsetzungen der kollaborativen Ausarbeitung. Wie auch unter 12.2 zu sehen ist, wurden zu diesem Zweck mehrere dedizierte Sitzungen mit ausgewählten Industriepartnern abgehalten.

Anfänglich wurde zu jeder erdachten Metainjektion ein Schlüsselwort thematisiert. In den ersten Sitzungen wurden somit vorerst, wie im Exposé ersichtlich, um die 20 verschiedenen *Kommandos* deklariert. Es entwickelte sich gedanklich zu einer Art Metakontrollsystem, einem Versionskontrollsystem in einem Versionskontrollsystem als Selbstverwaltungszweck, mit der Tendenz zur Unbeherrschbarkeit.

Schon bald wurde offensichtlich, dass die *inflationäre* Metaschlüsselwortvereinbarung, anstatt der geforderten Übersichtlichkeit und Eindeutigkeit, eher zu mehr Fehlinterpretationen durch Überschneidungen leitete. Die danach stattgefundene *Deflation* der *Schlüsselworte* sowie eine vereinfachte Syntaxform reduzierten den Gesamtumfang auf ein überschaubares und auch logisch nachvollziehbareres Maß.

Interessant war vor allem die Beobachtung, dass die Umstellung der Syntax von der ursprünglichen Kommandoform in eine Objektnotation mit mehr Freiheitsgrad in ihrer Anwendung, automatisch den zuvor vermeintlichen *ad hoc* Bedarf der definierten Schlüsselwörter erheblich minimierte und somit auch den Deflationsprozeß forcierte.

Die folgende Ausführung erklärt die derzeit verwendeten 4 Schlüsselwörter mit der in der Praxis am häufigsten genutzten Objekt- und Argumentensemantik. Es wird im Zuge der Implementationstechnikdarstellung, in Kapitel 9.3.2.a, nochmals auf die generische *Potenzialmenge* der möglichen Semantikabbildungen ausführlich eingegangen werden.

Um nicht den immer gleichbleibend strukturellen Aufbau eines jeden Pragmatismus erklären zu müssen, wird die Objektreferenz KAPITAL, das atomare Schlüsselwort **fett** und die Argumente je nach ihrem Bezug entweder als Atom *kursiv*, als referenzieller Bezug *KURSIV* oder als externer Verweis *#kursiv* dargestellt.

[1] Hier wird die Lernkurve in alternativer Definition als Lernaufwand betrachtet, siehe dazu http://www.sprachlog.de/2013/03/12/steile-kurven/

7.5.4 TASK.**state**(*start*|*stop*)

Das Atom state drückt den Status einer stationären oder auch nur temporären Beendigung beziehungsweise (Wieder)aufnahme der aktiven Tätigkeit auf dem Arbeitspaket TASK aus. Die Atome *start* und *stop* kennzeichnen den derzeitigen Tätigkeitszustand, durch ihre Injektion wird der zeitliche Verlauf gegeben durch den Zeitstempels des Commits verbindlich. Die jeweiligen start-stop Intervalle drücken somit einen *aktiven* Bearbeitungszeitraum des Arbeitspaketes TASK aus. Wird beispielsweise bei Commit 1 *start* vermerkt, Commit 2 & 3 bleiben unvermerkt, und erst bei Commit 4 *stop* gesetzt, so wird eine Zeitdifferenzberechnung zwischen den zwei legalen Zuständen mittels der Zeitstempel der Eincheckvorgänge vorgenommen.

7.5.5 TASK.**dependency**(*TASK*)

Dieser Pragmatismus dient der Darstellung einer, gegebenenfalls auch orthogonalen, Abhängigkeit. Damit ist gemeint, dass diese Information auch losgelöst von jeglicher kausalen Begebenheit injektiert werden kann, und in diesem Sinne eine faktische Ausdruckskraft besitzt. In der Regel wird dieses Schlüsselwort jedoch in Abhängigkeit diverser lokaler beziehungsweise arbeitspaketunabhängiger (globalen) Zustandsänderungen (state), oder auch in vereinbarten zeitlichen Synchronisationspunkten, oft benutzt.

7.5.6 TASK.**enquiry**(*#enquiryReference*)

Diese Methode besitzt den Charakter einer unidirektionalen ungerichteten Verbindungsanfrage. Die Idee ahmt eine Nachrichtenform der Web Services Description Language (WSDL[1]) nach, den Typ des *Solicit-Response*.

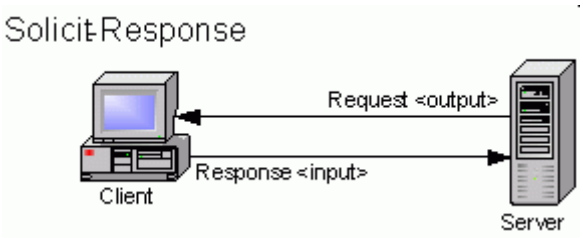


Abbildung 7.5 – Kommunikationsparadigma der *enquiry* Aktion

In Abbildung 7.5 übernimmt die Rolle des Clients hierbei RBMST, mit der Ausnahme, dass der Server ein nicht exakt adressierter Empfänger ist, deswegen auch ungerichtet. Das macht auch Sinn, denn die Anfrage wird somit in Richtung der Management-Hierarchie nach oben propagiert, bis zu derjenigen Instanz in deren Zuständigkeitsbereich sie fällt. Falls sie nicht beantwortet werden würde, stellt sich spätestens die Frage der Nichtbeantwortung, falls eine Eskalation dadurch entstünde – denn das Erstellungsdatum und die Referenz der verfassten Anfrage ist dem qualifizierten Publikum zugängig.

[1] http://www.w3.org/TR/wsdl

Der Nachrichtentransport auf diesem Medium, ist wie schon vorhin diskutiert, grundsätzlich unidirektionaler Natur. Das bedeutet, dass in diesem Fall der Server ein anderes Medium zur Beantwortung nutzen muss. Diese Gebarung ist grundsätzlich unternehmensspezifisch abzuklären, kann jedoch meist in Form einer dedizierten *Gruppenemail* schlüssig abgehandelt werden.

Es drängt sich noch die Frage nach der Art des formulierten Arguments auf. Prinzipiell ist die Argumentenstruktur *quasi frei definierbar,* die genaueren technischen Hintergründe werden in Tabelle 10.2 noch erläutert. Da aufgrund der hohen Flexibilität jedoch auch die Risikowahrscheinlichkeit einer Falschinterpretation steigt, sollte diese Art des Arguments im Sinne einer konventionellen Vereinbarung im Vorhinein festgelegt werden.

Es ist hier gängige Praxis, einen externen Verweis als indirekte Adressierungsart zu benutzen, darunter fällt eine ganze Klasse an Möglichkeiten. So könnte zum Beispiel der Verweis ein URL sein, ein vorgeschlagener Treffpunkt zur weiteren Diskussion oder auch eine eindeutige Referenz in einem Dokument, einem quasi offenliegenden Anfrageprotokoll. Letzteres kommt in der Praxis häufig zum Einsatz, da es einen dokumentarischen Charakter besitzt und dem Projekthandbuch als dynamisches Anfrage- und Besprechungsprotokoll beigelegt werden kann.

7.5.7 TASK.**knowledgeSource**(*#knowledgeSourceReference*)

Ein wichtiger Bestandteil jeglicher Entwicklungstätigkeit, worin die Software-Entwicklung sicherlich eine wichtige Untermenge darstellt, ist das Wissensmanagement. Es regelt die Schaffung und Aufbereitung von Wissen und Pragmatiken. Die Persistierung von Wissen ist wichtig, noch wichtiger erscheinen die Zugriffsart und die generelle Verfügbarkeit.

Falls es keine geeigneten Zugriffsmechanismen gibt beziehungsweise das Wissen über eine Vielzahl von Unterstützungssysteme verteilt, in fragmentierter Form darliegt, ist der erwartete Nutzen gering. Des Weiteren haben verwandte Arbeiten, wie sie im Kapitel 2.2.4 vorgestellt wurden, gezeigt, dass ein soziales Netzwerk ein vitales Element in der Software-Entwicklung darstellt, welches abgesehen von kulturellen Aspekten, unabdingbar für den Informations- und Wissensaustausch ist.

Das Primitiv[1] der knowledgeSource, bietet nun jener qualifizierten Gruppe ein einfaches Mittel um einerseits Wissensaspekte zu transportieren, andererseits auch in Bezug auf *soziales netzwerken* Informationen über neue Wissensartefakte zu propagieren. Die Hauptbenutzer von RBMST stellen prinzipiell die Entwicklungsteams dar, wobei die eigentlichen Nutzer die Managementteams sind. In diesem Fall jedoch dient RBMST, als Plattform der horizontalen Unternehmensebene, dem koordinierten Wissensaustausch.

Auch hier besteht die Möglichkeit, aufgrund des Freiheitsgrades der Referenzierung, URLs wie auch externe Dokumentenverweise zu nutzen. Falls das Unternehmen bereits etablierte Wissensmanagementsysteme betreibt, wird höchst

[1]ein eindeutiges minimales Mittel um einen Sachverhalt auszudrücken

wahrscheinlich die URL-Adressierung bevorzugt werden. Kleine Unternehmen verwenden auch häufig noch einfache Text-Dokumente zur einfachen Wissensabbildung [RU2004]. Unabhängig von der Referenzierungsart, ist das Faktum das die Beteiligten über diese Möglichkeit einen Propagierungmechanismus zur Verfügung gestellt bekommen, der das kreierte Wissen in engstem Zusammenhang mit der Problemstellung in der Software verknüpft, ohne den Quellcode direkt mit diesbezüglichen Kommentaren versehen zu müssen.

Das direkte Kommentieren von Wissensquellen im Code stellt, häufig mangels anderer Möglichkeiten, eine suboptimale Vorgangsweise dar. Bei nachträglichen Fehlersuchen ist diese Art der Information nicht sonderlich dienlich, da sie vorwiegend orthogonalen Charakter besitzt und somit vom eigentlichen Problem meist ungewollt ablenkt [MR2008]. Ein Extrahieren mittels Parsertechnologie um nachträgliches Wissen in eine Wissensdatenbank aufzubauen stellt ebenfalls eine Schwierigkeit dar, da der semantische Kontext im Quellcode nur in impliziter Form vorhanden ist.

Eine technische Alternative als Abhilfe zum Problem der unstrukturierten Quelltextkommentation bieten Werkzeuge wie *ScalaDoc*[1], *JavaDoc* und *DoxyGen*. Wobei die ersten zwei strikt auf Scala & Java ausgerichtet sind und das letzte einen eher generischen Ansatz vertritt [SM2008], bieten allesamt eine spezielle Kommentarsyntax an. Mittels dieser Notation ist es dem Entwickler möglich, ebenfalls in idiomatischer Form, Quellcode zu kommentieren um anschließend über die jeweilige Toolchain automatisch zu extrahieren und meist in Form von HTML-Seiten eine optisch ansprechende Dokumentation generieren zu lassen.

Diese Technik bietet zwei Vorteile. Einerseits ist es, durch die Idiomatik, für den Entwickler besser abgrenzbar, und andererseits bietet die automatische Dokumentationsausgabe ein Artefakt zur kontrollierten Verwaltung an, welches somit auch in geeigneter Form einem, der Entwicklung außenstehenden, Publikum ausgehändigt werden kann.

[1] http://docs.scala-lang.org/style/scaladoc.html

8 Kapitel

Die Funktionalität des Wirtsystems und die Auswahl einer Entwicklungsplattform

„Ich stelle fest, dass es zwei Wege gibt,
ein Software-Design zu erstellen,
entweder so einfach,
dass es offensichtlich keine Schwächen hat,
oder so kompliziert,
dass es keine offensichtlichen Schwächen hat.
Die erste Methode ist weitaus schwieriger.“
Tony Hoare

Schwerpunkte in diesem Kapitel:

- Der Implementierungsansatz
- Essentielle Implementierungsentscheidungen
- Die Festlegung des Entwicklungssystems

8.1 Der Implementierungssansatz

Die zuvor dargestellten Basismechanismen seitens des Versionskontrollsystems sowie der RBMSS-Schlüsselwortpragmatismen samt ihrer notwendigen konventionellen Vereinbarungen müssen auch mit den Sekundäranforderungen der Implementierung harmonieren. Nebenaspekte wie eine lose Systemkopplung und die graphische Eingabeschnittstelle stellen zwar keine Grundvoraussetzungen *per se* dar, spielen aber vor allem in Hinblick auf die erweiterte Einsatzfähigkeit und Usability ein essentielle Rolle in der Implementierung.

8.1.1 Die lose Systemkopplung

Die Implementierung verfolgt konsequent diesen Ansatz, da durch sie eine programmatische Gestaltungsfreiheit sichergestellt wird. Wie schon mehrmals angesprochen, ist der Fokus dieser Implementierung die Machbarkeit beziehungsweise Nutzbarkeit dieser These unter Beweis zu stellen.

Im Erfolgsfall kann dieser Systemrumpf als Basis für weitere Implementierungsschritte fungieren. Wie schon in der Softwaretechnik häufig erfolgreich angewandt, sollte die Kopplung der Module bezüglich Interdependenzen lose sein, jedoch die

„Kohärenz“ untereinander sehr dicht verlaufen [KD2004]. So kann auch das Prinzip auf die Interaktion, zwischen mehreren Anwendungen, erfolgreich übertragen werden.

8.1.2 Die Anbindung an das Repository

Das Wirtssystem Mercurial bietet aufgrund seiner Historie vielfältige Schnittstellen zur Interaktion mit dem zugrundeliegenden Repositorium an. Es kann kategorisch zwischen folgenden Anbindungsarten unterschieden werden.

8.1.2.a Der „builtin“ Hook Mechanismus

Diese Form der Interaktion erlaubt, selbst erstellte Funktionen an *neuralgischen* Punkten im System zu nutzen. Der Aufruf erfolgt zumeist mit von Mercurial vordefinierten Umgebungsvariablen, die der weiteren Informationsverfeinerung dienen. Die Hook-Granularität ist wie der gesamte Konfigurationsmechanismus an sich entweder global, sprich Repository unabhängig, oder auch lokal, Repository dediziert. Bei erfolgreicher Exekution wird der Wahrheitswert true statisch zurückgeliefert, andernfalls quittiert eine programmatische Ausnahme (Exception) die Fortführung. Alle Haken (Hooks) laufen im gleichen Prozesskontext von Mercurial ab. Die Einstellungen der jeweiligen Hooks werden in der Repository relevanten Konfigurationsdatei *.hgrc* vorgenommen, auf die Konfigurationssyntax wird jedoch nicht weiter eingegangen, siehe dazu[1].

Mercurial bietet in seiner derzeitigen Version knapp 20 Hooks für die verschiedensten Anwendungsanforderungen an. Der Übersicht halber werden nur die wichtigsten dargestellt und unverfälscht zitiert, um ein Vorstellung für das allgemeine Interaktionsspektrum zu bekommen.

— *commit*
Run after a changeset has been created in the local repository. ID of the newly created changeset is in "$HG_NODE". Parent changeset IDs are in "$HG_PARENT1" and "$HG_PARENT2".

— *incoming*
Run after a changeset has been pulled, pushed, or unbundled into the local repository. The ID of the newly arrived changeset is in "$HG_NODE". URL that was source of changes came is in "$HG_URL".

— *outgoing*
Run after sending changes from local repository to another. ID of first changeset sent is in "$HG_NODE". Source of operation is in "$HG_SOURCE"; see "preoutgoing" hook for description.

— *post-<command>*
Run after successful invocations of the associated command. The contents of the command line are passed as "$HG_ARGS" and the result code in "$HG_RESULT". Parsed command line arguments are passed as "$HG_PATS"

[1] http://www.selenic.com/hg/help/hgrc

and "$HG_OPTS". These contain string representations of the python data internally passed to <command>. "$HG_OPTS" is a dictionary of options (with unspecified options set to their defaults). "$HG_PATS" is a list of arguments. Hook failure is ignored.

— *pre-<command>*
Run before executing the associated command. The contents of the command line are passed as "$HG_ARGS". Parsed command line arguments are passed as "$HG_PATS" and "$HG_OPTS". These contain string representations of the data internally passed to <command>. "$HG_OPTS" is a dictionary of options (with unspecified options set to their defaults). "$HG_PATS" is a list of arguments. If the hook returns failure, the command doesn't execute and Mercurial returns the failure code.

— *pretxnchangegroup*
Run after a changegroup has been added via push, pull or unbundle, but before the transaction has been committed. Changegroup is visible to hook program. This lets you validate incoming changes before accepting them. Passed the ID of the first new changeset in "$HG_NODE". Exit status 0 allows the transaction to commit. Non-zero status will cause the transaction to be rolled back and the push, pull or unbundle will fail. URL that was source of changes is in "$HG_URL".

— *pretxncommit*
Run after a changeset has been created but the transaction not yet committed. Changeset is visible to hook program. This lets you validate commit message and changes. Exit status 0 allows the commit to proceed. Non-zero status will cause the transaction to be rolled back. ID of changeset is in "$HG_NODE". Parent changeset IDs are in "$HG_PARENT1" and "$HG_PARENT2".

— *listkeys*
Run after listing pushkeys (like bookmarks) in the repository. The key namespace is in "$HG_NAMESPACE". "$HG_VALUES" is a dictionary containing the keys and values.

— *tag*
Run after a tag is created. ID of tagged changeset is in "$HG_NODE". Name of tag is in "$HG_TAG". Tag is local if "$HG_LOCAL=1", in repository if "$HG_LOCAL=0".

— *update*
Run after updating the working directory. Changeset ID of first new parent is in "$HG_PARENT1". If merge, ID of second new parent is in "$HG_PARENT2". If the update succeeded, "$HG_ERROR=0". If the update failed (e.g. because conflicts not resolved), "$HG_ERROR=1".

Wie man anhand der obigen Liste erahnen kann, stellt die Parametrierung via Umgebungsvariablen, in Bezug auf die Plattformunabhängigkeit, einen gravierenden Nachteil dar. Meistens ist die Syntax plattformspezifisch, wie zum Beispiel bei MS-Windows, wo die Variablenevaluation mit dem ‚%' Prozentzeichen „geinfixt"

werden muss und im Falle von Leerwerten an das darüber liegende VCS erst gar nicht übergeben wird.

Der hieraus entstehende Datenfluss ist ausschließlich aktionsgetrieben. Das bedeutet, dass ein darauf basierendes Informationssystem in einer „snapshot“ Manier Daten zur Verfügung gestellt bekommt, die es in einem „ex ante“ - „ex post“ Bezug verarbeiten müsste, was sicherlich eine relativ komplexe Vorgehensweise dastünde. Diese Interaktionstechnik ist vorwiegend für kleinere Automatisierungsaufgaben [PK2013] per *Triggerausführung* konzipiert.

8.1.2.b Die Plug-In Unterstützung

Name	*Page*	*Description*
acl	AclExtension	Manage commit access to parts of a repo using control lists
blackbox	BlackboxExtension	Log events to .hg/blackbox.log for post-mortem debugging
bugzilla	BugzillaExtension	Update Bugzilla entries when a bug id is referenced in a changeset
churn	ChurnExtension	Show change statistics for mercurial operations per author
convert	ConvertExtension	Convert repositories from other SCMs into Mercurial
color	ColorExtension	Color output for the diff, status and qseries commands
eol	EolExtension	Translate line-ending characters between working copy and repository
extdiff	ExtdiffExtension	Compare changes using external programs
factotum	FactotumExtension	HTTP authentication with factotum
gpg	GpgExtension	Sign changesets and check signatures using GPG
hgcia	HgciaExtension	Send notifications to CIA
hgk	HgkExtension	Graphical repository and history browser based on gitk
highlight	HighlightExtension	Highlight syntax in the file revision view of hgweb
histedit	HisteditExtension	Edit, fold, drop changesets in the style of `git rebase --interactive`
keyword	KeywordExtension	Use CVS like keyword expansion in tracked files
largefiles	LargefilesExtension	Track large binary files (new in 2.0)
mq	MqExtension	Mercurial Patch Queues - manage changes as series of patches
notify	NotifyExtension	Send email to subscribed addresses to notify repository changes
pager	PagerExtension	Allows you to choose a pager
patchbomb	PatchbombExtension	Send a collection of changesets as a series of patch emails
progress	ProgressExtension	Show progress bars for some actions (new in 1.5)
purge	PurgeExtension	Purge all files and dirs in the repository that are not being tracked by Mercurial
rebase	RebaseExtension	Move revisions from a point to another
record	RecordExtension	Select working directory changes to commit by hunk à la darcs record
relink	RelinkExtension	Recreates hardlinks between repository clones
schemes	SchemesExtension	Add shortcuts to urls as url schemes
share	ShareExtension	Share repository history between multiple repos
transplant	TransplantExtension	Cherry-picking, rebasing and changeset rewriting
win32mbcs	Win32mbcsExtension	Allow to use shift_jis/big5 filenames on Windows.
zeroconf	ZeroconfExtension	Announce and browse respositories via zeroconf/bonjour

Abbildung 8.1 – die Mercurial PlugIns der Standarddistribution

Es gibt insgesamt um die 150, wobei ein kleinerer Teil von zirka 30 Erweiterungen von Mercurial offiziell gepflegt wird, diese sind in Abbildung 8.1 abgebildet. Mithilfe dieser Technik kann der Kommandosatz von Mercurial erweitert werden. Der Benutzer kann in seiner gewohnten Anwendungstechnik verharren, denn die neuen

Kommandos sind transparent integrierbar als wären sie bereits im Kern enthalten. Durch die Spezifikation der API werden gewisse Anforderungen an das PlugIn, um interoperabel zu sein, gestellt. Die Funktionalität ist ansonsten frei gestaltbar, und wird auch oft im Einklang mit der zuvor gezeigten Hook Technik verwendet.

Aufgrund dieser Flexibilität ist die Ausführung asynchron, in einem separaten Prozesskontext, ausführbar. Wegen der starken API Verzahnung ist für die PlugIn Erstellung Voraussetzung, dass die Programmiersprache Python, die die native Implementationsprache von Mercurial ist, verwendet wird. Die 3rd Party Extension sollte vor Veröffentlichung dem Mercurial CoreTeam zwecks Validierung übermittelt werden, da die internen API's auch fallweise überarbeitet werden müssen, und dementsprechend der PlugIn Code angepasst wird. Somit teilt sich die Gesamtmenge der Erweiterungen, in Mercurial gepflegte, Mercurial gehostet aber nicht aktiv gepflegt und in nicht öffentliche proprietäre PlugIn's auf.

Da die Entwicklung von PlugIn's den direkten Zugriff auf interne APIs[1] erfordert, ist ein harmonisieren der Lizenzbedingungen unerlässlich. Der Wirt ist quelloffen und bedient sich der GPL2+[2]. Diese Lizenzvereinbarung verlangt auch, dass Programme die sich der internen Schnittstelle bedienen ebenfalls unter die gleichen Lizenzgebarungen gestellt werden müssen [HM2004].

8.1.2.c Die Webserverfunktionalität

Wie auch bei einigen anderen VCS der Fall ist, bietet Hg ebenfalls eine Webschnittstelle an. Sie kann entweder indirekt über die CGI-Gateway Funktionalität eines in der Schicht darüber liegenden Webservices genutzt werden, oder auch über den integrierten leichtgewichtigen Webserver selbst. In jedem Fall bietet das Service einige Vorteile gegenüber der klassischen Kommandozeilenebene.

Einer der wesentlichsten Vorzüge stellt sicherlich, durch die standardmäßige Firewallfreigabe des Zielports 80, die geographische wie auch infrastrukturelle Entkopplung dar. Sind dateibasierte Operationen vorwiegend systemlokaler Natur können mithilfe des HTTP Dienstes nun Repositories dezentral konsolidiert gehostet werden, oder aber auch in transparenter netzwerkübergreifender Weise Entwicklungsartefakte komfortabel ausgetauscht werden. Die Frage der Authentifizierung verschiebt sich ebenfalls vom lokalen System hin zur Authentifikation via Webserver.

Ein weiteres Plus ist die ergonomische Bedienung des Systems und optische Darstellung der benötigten Informationen. Über pragmatische HTTP Verlinkungen von Lesezeichen über Etiketten bis hin zu Logs kann man mittels der Browsertechnologie schnell und zielgerichtet navigieren, ohne sich im Detail mit dem Kommandosatz des konkreten Versionskontrollsystems sich auseinandersetzen zu müssen – ein definitives Argument für Personen aus der abstrakteren Managementebene. Um das integrierte Webservice nicht unnötiger Weise im Sinne der gegebenen Bandbreite zu überlasten, wurde eine fokussierte Abbildung der gängigsten Kommandos auf HTTP-Ebene realisiert. Es ist somit auch nur ein kleines

[1] Application Prgrammer Interface

[2] GNU General Puplic License V2+

Spektrum der Möglichkeiten eröffnet, und in komplexeren Bedarfsfällen muss auf die Kommandozeile zurückgegriffen werden.

Dennoch besteht die Möglichkeit des benutzerdefinierten *Themings*. Hierbei nutzt Hg eine Technik die sehr leichtgewichtig und flexibel zu gleich ist. Durch ein zentrales Map-File pro Thema[1], gelangt man zu einen Template (Schablonen) gesteuerten Mechanismus, der letztendlich die über die Browseroberfläche angeklickten Elemente mittels der gewohnten Kommando Abfragestruktur die darzustellende angefragte Information extrahiert. Dadurch kann das System dem Nutzer dennoch das gesamte Kommandospektrum anbieten, welches er jedoch für seinen Bedarfsfall ergänzen oder adaptieren muss. Die Abbildung 8.2 zeigt das Standardthema des integrierten Webdienstes.

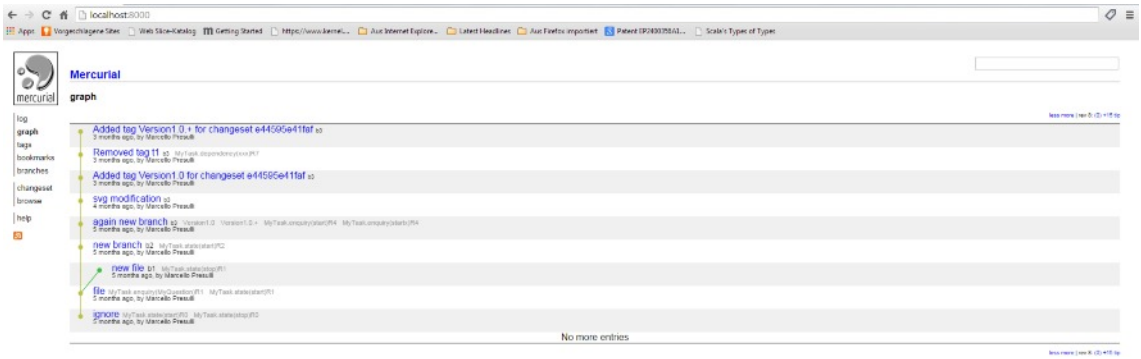


Abbildung 8.2 – das Standardthema des integrierten WebServers

8.1.2.d Die Kommandozeilenschnittstelle

Der gängigste Interaktionsmodus ist definitiv die Kommandozeile, durch diese Benutzerschnittstelle eröffnet sich das gesamte Nutzungsspektrum. Das „Look & Feel“ der Schnittstelle ist plattformunabhängig und wendet eine intuitive Syntax an. Die Kommandos sind durchgängig knapp aber schlüssig dokumentiert und durch einen Alias Mechanismus fällt der Umstieg von einem Fremdsystem durch eine bereits gewohnte Kommandonomenklatur leicht, da die Aliasing Technik eine transparente Übersetzung ermöglicht.

Es folgt generell dem Unix KISS-Prinzip[2], *Keep it small & simple*. Die Kommandos sind in ihrer prädestinierten Aufgabe beschränkt und intuitiv anwendbar. Natürlich spricht nichts gegen komplex Verkettungen mit den darunter liegenden Kommandointerpretern (Shell). Vor allem unixartige Shells wie die Zsh, Bash oder Korn Shell bieten hier ein reiches Repertoire an Kombinationsmöglichkeiten an. So sind komplexe Kommandoverkettungen wie

hg archive –type files –rev $Rev -I $(hg log -r$Rev –template {files} | sed 's/ / -I /g') ~/changedfiles

keine Seltenheit um die geänderten Dateien bezogen auf eine bestimmte Revision zu ermitteln. Solche Kommandosequenzen sind schon für technisch Versierte

[1] Theme ist gleichzusetzen mit dem dem optischen Web-Darstellungsthema

[2] http://de.wikipedia.org/wiki/KISS-Prinzip

herausfordernd, für Personen aus dem operativen Management und darüber naturgemäß fasst nicht bewerkstelligbar. Da die einzelnen Kommandos, im Gegensatz zur Webschnittstelle, geschrieben werden müssen, ist auch der verwendete Systemzeichensatz von tragender Bedeutung.

8.1.2.e Die Kommando-Server API

Die letzte Interaktionsmöglichkeit stellt der sogenannte Kommandoserver dar. Diese Art eröffnet, durch ihren einfachen aber effektiven Mechanismus, einige Möglichkeiten, die über die konventionellen Schnittstellen hinausgehen. Diese API ist seit Mercurial V1.9 bereits verfügbar und garantiert

— Lizenzrechtliche Unbedenklichkeit

— Stabilität

— Sprachneutralität

— Hohe Performance

Die gesamte Serverkommunikation erfolgt über den *low level stdin/stdout pipe* Mechanismus [DI2007], der auch oft bei SQL-Servern Verwendung findet. Hierbei wird die Standardausgabe des Servers mit der Standardeingabe des Clients verbunden, und umgekehrt. Die vereinbarte Byte Order ist *Big Endian*. Diese Form des Pipelinings ist in allen gängigen Betriebssystemen bereits seit den 1970'er Jahren klassisch vertreten.

Die Serverdaten werden in verschiedenen Kanälen, geprefixt durch ein ‚o'utput, ‚e'rror, ‚r'eturn oder ‚d'ebug ASCII-Zeichen, mit anschließender Datenlänge an den Client übertragen. Dieses Paradigma, hat den Vorteil, das die kommunizierende Client-Applikation keinerlei lizenzrechtliche Kompatibilitäten vorweisen muss, und bedingt durch die Einfachheit der Schnittstelle, ist der Client auch gegenüber Änderungen der internen API robust. Diese Stabilität ist für manche Anwendungen ein absolutes Muss, und gehen daher verstärkt den Weg über den Kommando-Server, da Sie das entwicklungstechnische Risiko einer etwaigen internen API Änderung nicht eingehen können, sei es aufgrund der proprietären Natur ihrer Anwendung oder aufgrund der Integrität der Daten.

Die Anbindung erfolgt wie bereits erwähnt über den klassischen Pipe-Mechanismus, eine Ressource die seitens des Betriebssystems zur Verfügung gestellt wird und somit auch sprachneutral ist. Die minimalen Anforderungen werden durch die API-Dokumentation ausreichend abgedeckt, wobei bezüglich Zeichensatzkompatibilität und Kommunikationsstörungsverhalten noch reichlich zusätzlicher Dokumentationsbedarf vorhanden ist.

Eine Besonderheit, wo diese Form der Kommunikation nicht betroffen ist, ist dass das Absetzen mehrerer Kommandosequenzen im Kommandozeilenmodus einen zusätzlichen Overhead der gesamten Applikationsinitialisierung generiert, der hier nur beim ersten Aufruf gegeben ist. Das stellt im sogenannten Batchbetrieb

des Kommando-Servers, also in der Stapelverarbeitung von Kommandosequenzen, einen signifikanten Verarbeitungsvorteil dar.

Aufgrund der einfachen Anbindung, gibt es aber auch Nachteile die es abzuwägen gilt. Etliche Konfigurationsparameter des Servers sind statischer Natur, also nach dem Startup nicht mehr änderbar, auch manche 3rd Party PlugIn Ausgaben die während der Betriebsphase anfallen entsprechen oft nicht dem Kommando-Server Protokoll und müssen behandelt speziell werden. Abbildung 8.3 verdeutlicht beispielshaft einen solchen Kommunikationsverlauf.

server	client	notes
	connected, waiting for hello message	
o: 52 capabilities: runcommand getencoding\n encoding: UTF-8		
		server is waiting for a command
	runcommand\n 7 summary	client talks to server on stdin
starts running command		
o: 27 parent: 14571:17c0cb1045e5		
o: 3 tip		
o: 1 \n		
o: 53 paper, coal: display diffstat on the changeset page\n		
o: 16 branch: default\n		
o: 16 commit: (clean)\n		
o: 18 update: (current)\n		
r: 4 0		server finished running command, writes ret on the 'r' channel to the client
	closes server stdin	client disconnects
server exits		client waits for server to exit

Abbildung 8.3 – beispielhafte Kommunikation des Command Servers

8.1.3 Die Extrahierungsarten der Nutzdaten

Die Diskussion der Schnittstellen eröffnet zugleich das Thema der Art und Darstellungsform der Daten. Die Vielzahl der angebotenen Schnittstellen suggeriert zugleich die Vorstellung einer Menge an Datenströme und ihre möglichen Formen der Kategorisierung.

8.1.3.a Der geblockte Datenstrom

Prinzipiell ist diese Transportform eine der Effektivsten. Sie wird im Kontext des Kommando-Servers verwendet liefert einen anfragegesteuerten Strom ohne jeglicher Vorinterpretation der Daten. Die auf den Daten basierende Information, sprich die lexikalische Analyse von Anfang und Ende eines relevanten Artefakts ist gerade bei dieser Transportart aufwändig. Eine der ersten Pilotversuche von RBMST nutzte diese Kommunikation und stieß, aufgrund der teilweise auch nicht vollständigen Dokumentation, diesbezüglich auf einige Probleme.

Unabhängig von dem im ersten Augenblick erhöhten Anreiz der Performance-optimierung ergeben sich dennoch Schwierigkeiten in Bezug auf ein zuverlässiges

Kommunikationsverhalten und etwaige Fehlerreaktionsmöglichkeiten. Da der Pipe-Mechanismus auf Betriebssystemebene realisiert wird, ergeben sich somit unterschiedliche Puffergrößen wie zwischen MacOS mit 16kB und Linux mit 4kB Standardgröße.

Des Weiteren ist auch die Performance zwischen den Terminal Standardein- und Ausgabe fraglich. Wird die Standardausgabe oft zeilengepuffert, hat man auch hier eine, ohne in den Source-Code eingreifen zu wollen, Performanceeinschränkung. Ein weiteres Problem stellt die Terminalinterpretation eines Zeilenendes dar. Bei manchen Endpunkten wird abhängig von den Terminaleinstellungen ein ‚\r\n' zu einem ‚\n' umgewandelt.

Auch die Erfassung des verwendeten Zeichensatzes ist entscheidend, da im Falle des Kommando-Server Kanaltransportes jeweils eine 4 Byte Längeninformation mitgegeben wird und diese Information ist, zum Beispiel im Fall von Kommunikationspartnern die mittels JVM Technologien realisiert wurden, behutsam auszuwerten da hier der Standardzeichensatz UTF-16, und nicht wie generell erwartet, 7bit-ASCII ist.

Bezüglich Kommunikationsstabilität muss seitens der Applikation Sorge getragen werden, dass zum normalen Evaluieren des Return-Wertes der Operation obendrein auch orthogonale Kommunikationsfehler der Pipe abgefangen und behandelt werden müssen. Die mittlere Bandbreite der Performance ist sicherlich hoch jedoch mit einer hohen Varianz verbunden, da die grundsätzliche Datensatzterminierung von Mercurial nämlich zeilenbasiert (\n) ist, und gerade diese Terminierungsart wird in anderen Subsystemen ebenfalls mitbehandelt und bringt deswegen auch kleinere Einbrüche in der Performance mit sich.

8.1.3.b Der eingebettete Template-Mechanismus

Hg lehnt sich bekanntlicherweise mit seinem Kommandoprinzip stark an das KISS-Prinzip an, dieses findet auch in der Schablonentechnik konsequente Verwendung.

Die Technik entspricht in etwa den konsekutiven Vorgängen des Formatierens, Weiterleitens (pipen) und Filterns von Prozessausgaben. Der Reiz dieser Technik liegt in der einfachen, gewohnten Anwendung und in seinem leichtgewichtigen flexiblen Ansatz. Rohdaten werden gemäß vordefinierter Schablonen gefiltert, sortiert und wenn notwendig auch geeignet konvertiert, ähnlich dem prozessieren in unixartigen Shells. Ein weiterer Vorteil ist, dass diese Implementation eine angebotene built-in Technik von Mercurial selbst ist, und somit auch stets weiterentwickelt und gewartet wird. Ein zusätzliches Anpassen bedeutet lediglich das Nutzen und kombinieren vordefinierter Primitive.

Das diese Technik eingebettet ist, ist auch die Performance erwartungsgemäß hoch anzusetzen, da Mercurial seine systembedingte Interna optimal auszunutzen versucht. Diese Templates decken einen Großteil der regelmäßigen Verarbeitungsmuster ab, und stoßen somit an Grenzen wenn speziellere Verarbeitungen notwen-

dig werden. Das selektive Filtern kann bezüglich einer Vorverarbeitung jedoch immer angewendet werden.

Zur Verdeutlichung zeigt das nachfolgende Beispiel die Extraktion des Benutzernamens samt den proprietären Mercurial Zeitstempel. Konvertiert in eine ISO 8601[1] Datums- Zeitangabe mit abschließendem Newline Zeichen (\n) als Zeilenende.

hg log -T "Name: {author} und Datum: {date|isodate}\n"

8.1.4 Lokalität der Information

Vor allem bei dezentralen Versionskontrollsystemen ist die Lokalität der Information aus der applikativen Sicht eine wichtiges Thema. Die Diskussion beginnt mit der kryptologischen Frage nach Authentifizierung und Sicherheit und endet bei der Ermittlung des durchschnittlichen Datentransfervolumens bei schmalbändigen Netzwerkanbindungen. Ein weiteres Kriterium stellt das dazugehörige Spektrum der Bedienungsmöglichkeiten mit dem korrelierenden Funktionsumfang dar.

8.1.4.a Lokales Dateisystem

Die klassische Persistierung der Information in Bezug auf Repositorien ist gewiss die Manifestation in einem lokalen Dateisystem. Es gibt ja bekanntlich eine Vielzahl von Dateisystemen mit den unterschiedlichsten Spezialisierungen. Der konkrete Einsatz hängt einerseits von den Anforderungen der Performance und Sicherheit ab, andererseits ist jene Auswahl zusätzlich beschränkt durch die Verfügbarkeit einer adäquaten Implementation für das verwendete Betriebssystem.

Generell unterscheidet sich die Datendurchsatzrate zwischen lokalem Dateisystem und einer durchschnittlichen Netzwerkanbindung zirka um den Faktor 1000. Dateisysteme mit kryptographischen „on the fly“ Verschlüsselungsmethoden sind etwas langsamer haben jedoch auch den unermesslichen Vorteil einer integrierten Sicherheit [KM2011].

Die Frage der Authentifikation wird im Regelfall eine Ebene höher geklärt. So sind Benutzersitzungen in Form von Anmeldungen über verschiedenste Authentifikationsformen heutzutage Domänen der Betriebssysteme geworden, die zusätzlich noch auf applikativer Ebene erweitert werden können.

8.1.4.b HTTP-Dienst

Dieses Service, das mittlerweile in die Jahre gekommen ist, hat letztendlich aufgrund der Anspruchslosigkeit weite Verbreitung gefunden. Der Dienst besitzt vom Protokollaufbau keinerlei Sicherheit und arbeitet prinzipiell dateiorientiert. Über die URL-Adressierung können Dateien des Repositoriums in Rohform betrachtet, beziehungsweise, auch heruntergeladen werden. Über eine übersichtliche Weboberfläche navigiert man durch sämtliche Rubriken wie Lesezeichen, Etiketten und Logs bis hin auf Dokumentenebene. Dieser Dienst der sich nicht

[1] http://de.wikipedia.org/wiki/ISO_8601

durch sonderliche Performance auszeichnet, ist jedoch bei visualisierten ad hoc Sitzungen, die eher den Ansatz einer einfachen Informationssuche verfolgen, sehr beliebt. In einem gesicherten Firmennetzwerk ist dem internen Gebrauch dieses Dienstes nichts entgegenzusetzen und auch empfehlenswert.

8.1.4.c Secure Shell Anbindung und HTTPS

Wie eigentlich der Name schon verrät, bietet das Wirtssystem eine gesicherte Form der konsolenbasierten Kommunikation (SSH) an. Dieses etablierte Protokoll kommt häufig bei Fernsitzungen zum Einsatz und hat mittlerweile seinen Vorgänger, das Telnet-Protokoll, größtenteils verdrängt. Das Protokoll erfordert aber keineswegs eine Fernsitzung, es können auch Lokalsitzungen, man könnte sie auch *localhost sessions* bezeichnen, abgehalten werden. Vordergründig ist hier vor allem die Notwendigkeit der Authentifikation.

Bei Fernsitzungen steht natürlich die Wichtigkeit der impliziten Verschlüsselung, mittels einem konfigurierbaren Kryptoalgorithmus, der Authentifikation in nichts nach. Auch gesicherte Portweiterleitungen lassen sich leicht realisieren, und können somit temporär und zuverlässig den impliziten Webdienst von Mercurial etablieren, da bis dato eine SSL-Implementation des built-in Servers nicht verfügbar ist.

Zur reinen Synchronisation zweier Repositorien kann jedoch eine HTTPS Verbindung, also HTTP über SSL, etabliert werden und somit ist die nachvollziehbare Authentifikation über X.509 Zertifikatsdienste[1] gewährleistet.

8.1.5 Normalisieren der Zeitbasen

Ein etwas anderes Problem stellt die über geographische Lokationen verteilte Software Entwicklung dar. In den letzten zwei Dekaden nahm das Outsourcen von Teilprojekten beziehungsweise das Offshoring[2] stetig zu. Die Hauptfaktoren sind kurz, ohne dieses Thema erschöpfend behandeln zu wollen, erklärt, die diametralen Lohnkosten, die Verfügbarkeit des Internets und der zunehmend kostengünstigere Betrieb von Netzinfrastruktur.

Ein Faktum, das sich jedoch immer wieder stellt, ist eine durch die Zeitzonen bedingte Arbeitszeitverschiebung [SE2001] und sich dadurch zwangsläufig ergebenden Zeitstempeln während des commits. Das Problem der sich teilweise überlappenden Zeitstempel muss von dem darüber liegenden System, in diesem Fall RBMST, behandelt werden. Es gilt dadurch abzuklären wie das Wirtsystem intern die Zeitbasen abspeichert und behandelt, um danach eine durchgängig zeittransparente Lösung zu entwickeln.

Exkurs Die Zeitzonen und ihre Synonyme

Geographisch bedingt, durch eine gleichmäßig über den Himmel laufenden

[1] http://de.wikipedia.org/wiki/X.509

[2] Bezeichnet die Verlagerung von Unternehmensbereichen ins Ausland

mittleren Sonne vorstellbar, muss die wahrgenommene Zeit durch einen zeitlichen Korrekturfaktor kompensiert werden. Teilt man die Erde in genau 24 gleiche Zeitabschnitte ein, so entstehen theoretisch 24 Zeitzonen. Diese erstrecken sich über je 7,5 Längengrade zu beiden Seiten des Bezugslängengrades, der im Ideal ein ganzes Vielfaches von 15° auf den international vereinbarten Nullmeridian ist.

Somit beträgt die Zeitverschiebung zu den benachbarten Zeitzonen ± 1 Stunde, mit einer mittleren Sonnenzeitabweichung an den Zonengrenzen von einer halben Stunde. Die meisten Staaten haben eine dieser Systematik entsprechend geregelte Zonenzeit, oder auch Zeiten falls das Staatsgebiet mehrere Zonen überdeckt, wie zum Beispiel Russland. Über die Zeit haben sich auch verschiedene Nomenklaturen entwickelt wie die *GMT* synonym zu *UTC* und auch zur militärischen *Zulu* Zeitzone ist, also dem Zeitzonennormal mit der Abweichung 0. Es gibt noch viele, historisch bedingt weitere Synonyme, siehe dazu [1].

8.1.5.a Zeitrepräsentationen

Die Basis UTC bildet den Standard jeglichen Zeitabdruckes im Versionskontrollsystem. Das Wirtsystem erfasst den „lokalen" Zeitstempel relativ zur Zeitzone des Repositoriums wo das Changeset manifestiert werden soll. Somit ergibt sich eine proprietäre Notationsform, die Mercurial Zeitrepräsentation.

Es ist ein Zeitstempel der Unix-Epoche, sprich die fortlaufende Sekundenanzahl beginnend seit dem *01.01.1970 0:00* UTC, anschließend durch einen Punkt ‚.' getrennte Stelle einer Dezisekunde die jedoch immer Null (0) ist. Danach folgt entweder eine negative Zahl beginnend mit einem Minuszeichen (-) oder eine vorzeichenlose positive Zahl, beide sind Zahlen des Sekundenbereichs. Negativ dann, wenn die Zeitzone östlich von UTC liegt, positiv falls westlich. Ein Beispiel soll die Ausführung verdeutlichen.

Der Zeitstempel *1398412524.0-7200* drückt durch den negativen Wert -7200 aus, dass der vordere Teil also die *1398412524* Sekunden um die 7200 Sekunden verringert werden muss um auf die normalisierte Zeit zu kommen. -7200 Sekunden entsprechen somit der östlichen Zeitzone und zwar um 2 Stunden verschoben, somit ist die geographische Zone etwas eingegrenzt.

Verkompliziert wird das Ganze noch aufgrund etwaiger Sommerzeitverschiebungen (DST[2]). So kann der obige Zeitstempel eine mitteleuropäische Sommerzeit wie auch beispielshaft eine südafrikanische Zeit ausdrücken, da Südafrika auch in der Zeitzone 2+, jedoch ohne Sommerzeitverschiebung, liegt.

Somit ist diese Form der internen Repräsentation zwar kompakt und deterministisch jedoch ohne weiteren Bezug einer verbal formulierten Zeitzone wie *EET* für East European Time oder *SAST* für South African Standard Time für eine geographisch eindeutigere Zuordnung zu wenig ausdrucksstark.

---

[1] http://de.wikipedia.org/wiki/Zeitzone#Liste_der_Zonenzeiten

[2] Daylight Saving Time

8.2 Essentielle Implementierungsentscheidungen

Dieses Unterkapitel dient zur Bestimmung und Festlegung verschiedenster Vorgaben, auf die das künftige System RBMST basieren soll, um den gewünschten Nutzen zu erzielen. Die generelle Strategie die hier verfolgt wird, wurde unter Kapitel 8.1.1. bereits kurz angesprochen – eine lose Systemkopplung. Diese Forderung ergibt sich aufgrund zweier Maxime. Die Maxime der erleichterten Austauschbarkeit des darunterliegenden Wirtsystems sowie die weitgehende Plattformunabhängigkeit.

Die erste Bedingung kann nur dann gewährleistet werden, wenn die Schnittstellenverzahnung gering gehalten wird. Es gilt somit eine Schnittstelle zu nutzen die in den meisten Versionskontrollsystemen in ähnlicher Weise vertreten ist. Die zweite Forderung adressiert die Art und Weise der entstehenden Applikation. Hier sollte vor allem der Ansatz einer virtuellen Maschine verfolgt werden, da diese im Gegensatz zu nativen Übersetzungen wie beispielweise bei C++ keine harte Maschinencodeanbindung zum Prozessorkern besitzen, sondern viel mehr den Weg über eine teilweise emulierte Laufzeitumgebung (Runtime Environment) gehen.

8.2.1 Schnittstellenbestimmung

Die Frage nach der Wahl der geeigneten Schnittstelle hängt von vielen Faktoren ab, die es zu gewichten gilt. Ein wichtiges Argument hierbei ist der benötigte Datendurchsatz. Ist die gedachte Applikation durch ihre Anforderungen sehr datenintensiv sollte im Konkreten die Kommando-Server Schnittstelle bevorzugt werden. Im Fall von RBMST hingegen besitzen die anforderten Daten den Charakter prägnanter Metainformationen. Hier geht es weniger um einen Nutzdatenverkehr, wie beispielsweise bei einen commit von 100 Dateien, sondern vielmehr um Abfragen von Zeitstempeln, Autorennamen und natürlich auch die Informationen der injektierten Schlüsselpragmatiken selbst.

Ein weiterer Aspekt ist die lose Systemkopplung. Auch hier sind Techniken wie Hooks, Plug-Ins und der Kommando-Server eher aufwändiger. Hooks sind hinsichtlich der technisch bedingten Anwendungsform für RBMST suboptimal. PlugIn's hingegen würden sich zwar eignen, sind jedoch aufgrund der nicht vorhandenen Sprachneutralität und der starken Verzahnung mit der internen API des Wirtsystems zu plattformgebunden. Ein dritter Punkt, der bezüglich der Nutzbarkeit genauso wichtig ist, ist die Möglichkeit auch transparent, netzwerkübergreifende Schnittstellen der Versionskontrollsysteme zu nutzen. Die Bedienung rein lokaler Systeme stünde hier anachronistisch zur realen Welt verteilter Software Entwicklung.

Der Aspekt der Authentifizierung spielt hier aus Sicht von RBMST definitiv eine untergeordnete Rolle, das System setzt hier eine bestehende Realisation sei es über SSH-Porttunneling oder ein dediziertes VPN[1] voraus. Wie in Tabelle 8.1 dargestellt,

[1] Virtual Private Networking

ergibt sich somit eine mehrdimensionale Entscheidungsmatrix, die die verfügbaren Techniken in Bezug auf die obigen Systemanforderungen gegenüberstellt.

Schnittstelle	*Lokalität*	*Systemkopplung*	*Performance*
Hooks	lokal	verzahnt	ungeeignet
Plug-Ins	lokal	verzahnt	geeignet
builtin WebServer	lokal/fern	lose	ungeeignet
Kommandozeile	lokal/fern	lose	geeignet
KommandoServer	lokal	lose	geeignet

Tabelle 8.1 – Gegenüberstellung der unterschiedlichen Betriebsarten

Um eine rationale Entscheidung treffen zu können werden die Bewertungskategorien kurz vorgestellt, um auch im Detail differenziert werden zu können. Die Unterscheidung lokal/fern stellt hier nur die klassischen Anwendungsfälle dar, so ist es im Fall des Webservers natürlich auch möglich eine lokale Instanz auf der Workstation selbst zu nutzen. Im anderen Fall bei der Kommandozeilenschnittstelle ist über eine geeignete SSH-Sitzung natürlich auch ein fernes operieren möglich.

Die Systemabhängigkeit wird hier durch das Wortepaar verzahnt/lose ausgedrückt. In diesem Fall ist die Bezeichnung lose beim Kommando-Server ebenfalls nicht ganz eindeutig, da zwar der Pipe-Mechanismus einerseits, aber andererseits die Interpretation des Zeilenendes und anderer Kontrollzeichen der Standardein- und Ausgabe eine gewisse Plattformabhängigkeit darstellen [LD1994]. Die lose Systemkopplung kann ebenfalls eine einfache transparente Zeichenstrombehandlung von HTML-Textdokumenten, und nicht etwaige verbindungsspezifische HTTP-Request/Response Techniken, voraussetzen.

Dasselbe gilt für das Performance clustering ungeeignet/geeignet. Hier wird nur das Leistungsmerkmal im Sinne von Nutzdatentransfers verstanden. Die Anforderungen seitens RBMST könnten, zwar abhängig von der Dichte der genutzen Schlüsselwortpragmatiken aber dennoch prinzipiell, von jeder Schnittstelle geleistet werden. Eine Ausnahme stellen dabei die Hooks dar, denn bei diesem Technikeinsatz wäre schon die Anwendungskomplexität nicht angemessen.

Um die Vorteile der lokalen wie auch fernen Bedienbarkeit, unter besonderer Berücksichtigung der losen Systemkopplung, Portabilität und der Stabilität der Kernfunktionalität zu nutzen, wurde die Kommandozeilen- wie auch die HTTP Schnittstelle zur Anbindung an das Wirtsystem erkoren.

8.2.2 Robustheit & Systemoffenheit

Diese beiden Begriffe knüpfen indirekt an die zuvor geführte Diskussion der Schnittstellenauswahl an, indem die Forderung der Losigkeit im Bedienen und Konfigurieren aus Sicht der Applikation logisch fortgesetzt werden muss. Da ansonsten aus der obersten Applikationsschicht, der Benutzerschnittstellen, eine zu starke Abhängigkeit in den syntaktischen Feinheiten der Wirtsysteme vorherrschen würde.

8.2.2.a Vorteile und Nachteile des lexikalischen Parsens in RBMST

Eine der ersten Vorstudien zu RBMSS widmete sich unter anderem dem Einsatz von Grammatiken, speziell in Bezug auf DSL. Es stellte sich jedoch nach reiflichen Überlegungen sehr bald heraus, dass einerseits eigentlich diese Anforderung gar nicht die Notwendigkeit einer eigenen domänenspezifischen Sprache verlangt und andererseits aufgrund der heterogenen Zusammensetzung der Benutzerschaft auch obendrein nicht unbedingt verlangt wird.

In den kollaborativ abgehaltenen Sitzungen kristallisierte sich der Einsatz geeigneter Scannertechniken mehr und mehr heraus. Im Gegenteil, desto mehr Beteiligte sich mit der Aufgabestellung der Informationsabbildung befassten, umso mehr wurde eine kompaktere Form zur programmatischen Behandlung entwickelt. Das gesamte Szenario erfuhr eine positive Eigendynamik wo allesamt einen *common sense* in ihrer Vorstellungswelt entwickelten. Um im weiteren Verlauf die richtige Wortschärfe zu finden, soll ein kleiner Exkurs dazu abgehalten werden.

Exkurs Scanner & Parser

Diese beiden Instanzen werden oft synonym gebraucht, besitzen dennoch eine andere Zweckwidmung beziehungsweise Einsatzbandbreite.

Das *Scannen*, die konkrete Implementation eines lexikalischen Suchvorganges, stellt eine Art Vorstufe des *Parsens* dar. Es befasst sich mit der lexikalischen Zerlegung von Zeichenströmen, und stellt somit eine atomare, auf Zeichensatzebene basierte Aufspaltung eines Datenstroms dar. Das Resultat entspricht der Zerlegung auf Tokenebene[1], also Wörter die eine indentierte Syntax besitzen sollen, in anderen Zusammenhängen auch als Schlüsselwörter oder Primitive genannt. Wichtig dabei sind Angaben eines regulären Ausdrucks über ein vereinbartes Alphabet, also jener Zeichen die ein potenzielles Schlüsselwort der Syntax ergeben können. Aber auch Operatoren und Separatoren, sprich Trennwörter, jener Zeichenvorrat der eine Aneinanderreihung von Wörtern zu einem syntaktischen Ausdruck, einen Satz, ermöglicht.

Der Parser basiert auf dieser Vorarbeit, und nimmt dann seine Aufgabe gemäß des dargelegten syntaktischen Ausdrucks auf. Er wendet Regeln der Grammatik an um die Validierung eines gültigen Ausdrucks zu vollziehen. Diese Technik wird sehr oft mittels einer Baumstruktur realisiert, dem Abstrakten Syntax Baum, kurz

[1] http://de.wikipedia.org/wiki/Token_(%C3%9Cbersetzerbau)

AST, wo die Beziehungen zwischen Elternknoten und Kindern kompakt abgebildet werden können [LK2012b].

Diese Technik ist deterministisch und wurde von den Implementatoren bereits gut verstanden. Die Parser-Instanz stellt somit die Schicht über der Scanner-Phase dar und richtet sich auf eine logische Überführung der Token in eine zielsprachabhängige Ausdrucksvalidierung. Die dann letztendlich die semantische Natur der gedachten Formulierung, in entsprechenden Maschinencode, transformiert.

Exkurs Ende

Ein sequentielle lexikalische Behandlung besitzt, wie man sich bereits vorstellen kann, einen hohen Grad an Effizienz in Bezug auf eine deterministische Analyse. Diese Technik ist somit relativ robust und anspruchslos im Verhältnis zum Parser. Anspruchslos deswegen, denn es werden nur sehr wenige Regeln zur Token-Bestimmung verwendet und im Fall einer Negativbestimmung die analysierte Teilzeichenkette bis zum nächsten Separator einfach verworfen, und von dort an weiter fortgesetzt. Robust, da Algorithmen wie Backtracking, Lookahead oder Ähnliche nicht zur Anwendung kommen. Nichts desto trotz ist das Parsen und die Definition von Grammatikregel für eine korrekte Sprachbehandlung unerlässlich und notwendig, dafür bieten sich jedoch eigene Sprachgeneratoren wie der zuvor schon erwähnte ANTLR an.

Die Schlüsselwortpragmatismen stellen im Sinne einer Informationsklassifikation lediglich den Anspruch an eine lexikalische Analyse, die sich auf reguläre Ausdrucksmuster beschränkt. Es werden somit aus Datenströmen *Informationstoken* gewonnen, die in höherer Ebene nicht als grammatikalische Sätze, sondern viel mehr als eine logische Aneinanderreihung faktischer Informationsverläufe behandelt werden.

Wie schon in dem Unterabschnitt 7.4.3 angedeutet wurde, bedingt diese technische Lösung jedoch auch organisatorische Konventionen, oder zumindest entsprechende Hilfsapplikationen. Denn im Fall einer typographischen Fehleingabe der gedachten Token, wird das illegale Muster einfach verworfen und die Verarbeitung fortgeführt. Das entspricht jedoch nicht der grundsätzlichen Intention, es deswegen auch eine geplante Überwachung der eingegebenen Metainformationen stattfinden. Sei es eine Art der manuellen Überprüfung die jedoch ebenfalls unzuverlässig sein kann, oder eine programmatische Abhilfe mittels einer kontrollierten Eingabemaske die die Fehleingaben im Vorfeld abfängt und kommuniziert.

8.2.2.b Offenheit durch Minimalitätsanforderung

Eine weitere Betrachtung auf der anderen Seite der Applikationsebene behandelt die minimalen Anforderungen seitens der statischen Konfiguration beziehungsweise der Parametrierung der Bedienerschnittstellen. Um den Wünschen der Anwender nach Ergonomie und Flexibilität zu entsprechen, bietet RBMST eine Vielzahl an Schnittstellen an. Es wird noch in Kapitel 10.2 auf die Eigenheiten der einzelnen Interaktionsmöglichkeiten konkret darauf eingegangen werden. Wichtig

in diesem Kontext ist, einen gemeinsamen Nenner in der Minimalanforderung des darunterliegenden Wirtsystems samt seiner benötigten Schnittstellenparameter zu finden. Somit ist einerseits klar, dass zum Beispiel die Parametrierung der Webschnittstelle eine andere sein wird als die für das lokale Dateisystem. Dennoch kann man durch eine geschickte Überführung die minimale Abweichung der dazu benötigten Parameter erarbeiten. Die Konfigurationsdatei der Stapelverarbeitung von RBMST zeigt in Abbildung 8.4 anschaulicherweise ein minimales Parametrierungsszenario auf.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<Configuration>
<DotPath>C:\Program Files\Graphviz</DotPath>
<OutputPath>C:\Users\pruefstand\Desktop\output</OutputPath>
<!--Project Configurations-->
      <Project RepoPath="http://localhost:8000/"/>
      <Project RepoPath="C:\Users\Pruefstand\Desktop\dummy"/>
</Configuration>
```

Abbildung 8.4 – Grundlegende Parametereinstellungen via XML-Tagging

Dieses XML-File stellt die prinzipiellen Minimalanforderungen an den Benutzer dar. Wobei aus der informationsverarbeitenden Sicht die Angaben einer indentierten Metainformationsextraktion fehlen, zeigt diese Konfiguration 3 unterschiedliche Parameter, wobei die ersten zwei seitens RBMST zur Speicherung und Visualisierung benötigt werden. Der einzige zwingende XML-Tag in Bezug auf das Wirtssystem ist somit der <Project> Tag mit seinem Attribut RepoPath, das wie man sehen kann entweder ein HTTP gemanagtes oder auch auf einem lokalen Dateisystem gelegenes Repository adressieren kann. Die Unterschiede und Anforderungen in der Parametrierung sind damit auf das Minimale beschränkt, und ebnen den Weg eines potentiellen Austausches des Wirtsystems und ist somit bezüglich neuer Anbindungen offen. Gleichzeitig ist durch die Parametrierung auf Dateipfadebene eine Repositoryanbindung über ein netzwerkfähiges Dateisystem, kurz NFS, ebenfalls denkbar, sollte aber in Bezug auf Performanceaspekte vor einem geplanten Produktiveinsatz eingehend evaluiert werden.

8.2.2.c Die Stabilität der Kernfunktionalität des Wirtsystems

Das in der Implementierung verwendete Wirtsystem Mercurial hat seit seiner Entstehung 2005 bis dato reichlich an Funktionalität dazugewonnen, derzeit befindet sich das VCS in der Veröffentlichungsversion 3.1.1. Bis dahin wurden viele Plug-Ins und Workflows entwickelt um die Vision von dezentralen Versionskontrollsystemen zu verwirklichen. Einige von ihnen wurden entweder vom Mercurial Core-Team zur gepflegten Wartung übernommen, oder sogar in die Kernfunktionalität mit aufgenommen. Eine dieser Erweiterungen ist zum Beispiel die seit dem Jahr 2008 bestehende BookmarkExtension, eine Idee vom Mercurial Schöpfer Matt Mackal selbst welche seit langem ein fixer Bestandteil des Systems ist.

Obgleich sich in den verschiedensten VCS teilweise die Schnittstellenaufrufe, die Datenbankanbindung beziehungsweise auch das interne Datenformat anfänglich mehrmals änderte, bildete jedoch eine gewisse Stammfunktionalität das Kernstück, welches sich nur spärlich bis garnicht änderte. So stellte auch wie bei vielen anderen Entwicklungen die Kommandozeile den Anfang dar und endete schlussendlich mit graphischen Oberflächen und Funktionalitäten reiner Bequemlichkeitszwecke wegen. Ein hohe Varianz in der Stammfunktionalität ist somit nicht mehr gegeben, der Kern hat sich mittlerweile stabilisiert.

Im Wesentlichen gleichen sich sämtliche Vertreter der Gattung DVCS[1]. Ohne jetzt auf die Semantik der einzelnen Kommandos detailliert eingehen zu wollen, werden beispielhaft die gängigsten kurz aufgeführt. Es ist dabei wichtig zu bemerken, dass ein hoher Prozentsatz der Kommandoschlüsselwörter entweder per se in den jeweiligen Vertretern integriert ist, oder aber über den bereits an anderer Stelle vorgestellten Alias Mechanismus zur Verfügung gestellt werden, und somit einen vereinheitlichten Kommandosatz zu haben um dem potenziellen Nutzer einen Umstieg zu erleichtern. Bei den am häufigsten verbreiteten Vertretern wie Git, Mercurial und Bazaar sind mindestens die unten angeführten Kommandos in ihrer Syntax und Grundsemantik ident.

— c*ommit*, Manifestation des Changesets auf Repositoryebene

— l*og*, Visualisierung des Kommentare während der commit Phase

— p*ush & Pull*, synchronisieren der Daten mit einem verwandten Repository

— b*ranch*, aufspalten eines Hauptstranges in Unterverzweigungen

— m*erge*, kontrolliertes Zusammenführen mehrerer Zweige zu einem Strang

Wie man hierbei erkennt, gibt es de facto keinen Unterschied in der syntaktischen Form, auch die Semantik ist größtenteils ident, wobei bei einer etwaigen Portierung auf subtile Unterschiede exakt eingegangen werden sollte.

Größeres Augenmerk muss jedoch auf den Bookmark-Mechanismus gelegt werden. Diese Referenztechnik, die als abstrakte Zeigerebene über den Changesets liegt ist bei Mercurial sehr volatil und effizient gelöst [LS2009]. Sie entspricht analog dem „Git Branch“ beziehungsweise „Git Ref“ Schema, in *Bazaar* kann dieser Mechanismus ebenfalls über Branchtechniken emuliert werden. Es werden hier jedoch Datenfragmente unveränderlich im Repository abgelegt. Damit ist diese Technik auch verhältnismäßig *teuer*. Die Forderung nach Seiteneffektfreiheit, der Daten des Repositoriums, wäre auch hier nicht gänzlich erfüllt.

Ein generelles Positivum stellen die möglichen Konvertierungspfade dar. Die hier oben angeführten SCM bieten allesamt Möglichkeiten zum bidirektionalen Überführen in das jeweilige Zielrepository an. Mercurial, stellt beispielsweise mit dem Kommando *hg convert* in einfacher Art und Weise Konvertierungseinstellungen parat, die einen transparenten Import von Git, Bazaar, Subversion und Perforce Repositories ermöglichen [MC2013].

[1] Distributed Version Control System

Aber auch hier muss behutsam vorgegangen werden, da die Imports nicht über alle erdenklichen Vorversion der jeweiligen Fremdrepositories bedenkenlos eingesetzt werden können. Im Bedarfsfall bietet sich jedoch im Vorfeld ein *Migrationspfad* durch Versionsaktualisierung des zu importierenden Fremdversionskontrollsystems an.

8.2.3 Die RBMST Ausgabeschnittstellen

Da die Schnittstellenauswahl zur Interaktion mit dem Repository über das Wirtsystem nun bestimmt ist, ist die anderer Seite der Applikationsschicht, nämlich die Ausgabeseite, der Output, ebenfalls zu diskutieren. Die ausgabeseitige Schnittstelle besitzt einen etwas anderen Fokus als ihr Pendant. Standen bei den Eingabeschnittstellen die einfache Anbindung, die Vielseitigkeit und die Performance im Vordergrund stellen hier die Prämissen der einfachen Darstellbarkeit und der potenziellen maschinellen Weiterverarbeitung die Schwerpunkte dar. Die Bestimmung dieser Mensch-Computer-Interaktion, kurz HCI, ist ein essentieller Faktor der in gewisser Weise den Endnutzen darstellt, und dieser bestimmt in weiterer Folge die Einsatzhäufigkeit.

Falls die Software (RBMST) seitens der Lizenzbestimmungen den Weg des „open source“ Modells einschlägt und die Benutzungshäufigkeit eine kritische Masse überschreitet, ist die Wahrscheinlichkeit gegeben, dass eine Community die Entwicklung dieses Werkzeuges weiterführt und somit zu einer anderen Reifeform hinsichtlich Stabilität, Funktionalität und Portabilität bringen könnte.

8.2.4 Die Wahl der Notations- & Serialisationsform

In den Anfangsstadien von RBMST wurde eine proprietäre Darstellungsform verwendet. Diese diente hauptsächlich im Zuge der Entwicklung als Debug-Ausgabe und wurde stetig kompakter und eindeutiger in ihrer Ausdruckskraft. Einige meiner Kollegen fanden sie sogar intuitiv und angemessen. Sie verwendete einfache ASCII-Zeichen um eine Zeigerdarstellung zu realisieren, mit der Relationen abgebildet werden konnten.

Es war somit eine einfache Notationform zur Graphenserialisation mittels einer Teilmenge des ASCII-Alphabets entstanden, die sich rudimentär an die Darstellungstechnik der Mercurial Erweiterung *GraphlogExtension*, die mittels *hg log -G* aufrufbar ist, anlehnte. Ein kurzes Beispiel in Abbildung 8.5 soll die Syntax dieser Serialisationsform dokumentieren.

$$\{http://mybook.com\}[Autor] -> "Joe\ Doe", [Seiten] -> "256"$$

Abbildung 8.5 – selbst entwickelte Pseudosyntax zwecks Graphennotation

Jede Ressource, im S-P-O[1] Jargon, jedes Subjekt und Objekt wurde durch das Klammernpaar {} gekapselt, ein Prädikat hingegen mit dem Klammernpaar []

[1] Subjekt-Prädikat-Objekt Aussage, die die Beziehungen untereinander ausdrückt

geklammert verwies mittels Pfeilsyntax -> an das Objekt, welches auch als untypisiertes Literal in Anführungszeichen dargestellt werden konnte. Falls ein Subjekt mehrere Prädikat - Objekt Beziehungen aufwies, wurden sie in einer Listenform durch Kommata , getrennt. Diese Form wirkt auf den ersten Augenblick tatsächlich intuitiv, hat aber nur begrenzte Aussagekraft, so wurden Verschachtelungen, wie zum Beispiel Objekte die als Ressource weitere Attribute besitzen können sehr unhandlich.

Als diese Notation in den kollaborativen Sitzungen als eine mögliche Darstellungsform präsentiert wurde, änderte sich der weitere Verlauf der Ausgabeformatierung. Mehrheitlich bestand zwar die Tendenz zu einer textuellen Notationsform, aber maximal als Austauschformat zwischen Systemen, beziehungsweise als Basis für eine potenzielle Weiterverarbeitung. Ein klarer Trend setzte sich zugunsten einer graphischen Repräsentationsform durch. Somit war klar geworden, dass diese Notationsform maximal als ein *intermediate Source-Code* Format gedient hätte, dass demnach proprietär weiterprozessiert hätte werden müssen um eine graphische Darstellung daraus zu generieren. Eine standardisierte Technologie war von Nöten um diesen Prozess geradlinig abdecken zu können. Einen konzeptionellen Einfluss auf eine zweckdienliche Darstellungsform lieferte die Arbeit *„A Semantic Web approach to Digital Rights Management“* [GR2005], die Idee der Abbildung von RDF auf SVG war somit initiiert.

8.2.4.a RDF/XML

Diese Ressourcendarstellung erfüllt die Notwendigkeiten und eignet sich optimal, implizit durch ihre Charakteristik, zur Darstellung der verschiedensten Informationsfragmente. Diese werden durch die Schlüsselwortpragmatiken injektiert, und stellen prinzipiell eine Aussage der Form S-P-O dar ohne eine eigene konkrete Ontologie darzustellen. Wie schon in Kapitel 6.3 dargestellt, kann durch diese Analogie RDF sehr gut angewandt werden, da sie eine Schicht unterhalb der Ontologiesprachen darstellt, und dadurch als Basisträger im semantic Web den de facto Standard darstellt.

Es soll jedoch nicht unerwähnt bleiben, dass die Beschreibung mittels Topic Maps (XTM) und anschließender Serialisierung über XML ebenfalls eine taugliche Alternative darstellen würde. Die Verfügbarkeit der Werkzeuge zu einer adäquaten Weiterverarbeitung ist jedoch nicht so stark verbreitet, wobei die Ausdrucksstärke der Notation an sich, den Anforderungen mehr als entsprechen würde [PJ2003].

In Kombination mit dublin core, vcard und dem selbst erstellten Namensraum RBMSS ist die holistische Darstellung durch die eklektische Referenzierung der konkreten Namensräume durchgängig möglich. Durch die starke Verbreitung von RDF als Austauschformat mit Namensraumbezug eröffnet sich eine Bandbreite verschiedenster Werkzeuge, die dieses Format weiterverarbeiten können. Die häufige Nutzung im Internet und die Fokussierung auf Austauschbarkeit, zieht die Serialisierung in XML den anderen Notationsform, wie Turtle/N3 vor, wobei letztere für den Menschen sicherlich intuitiver wäre. Eine von der Stanford Univerity,

entwickelte und sehr beliebte häufig verwendete Toolsuite, namens Protégé[1], kann beide Alternativen problemlos verarbeiten.

8.2.4.b Scalable Vector Graphic

Die tatsächliche gewünschte Ausgabeform soll grafischer Natur sein. Es stellt sich somit die Frage mit welchen Inferenzierungsprozess kann nun jene Information abgeleitet werden um dementsprechend eine zuverlässige Abbildung der Knoten, Kanten und Literale aus dem RDF-Graphen zu gewährleisten. Hier eröffnet sich der klassische Weg über die verfügbaren Technologien der XML-Familie. Das standardisierte Prozessieren über einen XSLT[2] Prozess garantiert eine eindeutige Überführung in das gewünschte Zielformat. Eine Frage bleibt noch offen, nämlich in welchem graphischen Ausgabeformat die Konvertierung durchgeführt werden soll. Hierzu gibt es zwei Anforderungen, erstens soll das Ausgabeformat einen offenen Standard genügen, und zweitens soll das Format mit Standard-Toolkits visualisier- und bearbeitbar sein. Die Forderung der Bearbeitbarkeit entstand indirekt, da in den iterativen Sitzungen auch die Frage nach Möglichkeiten der Adaption und der Navigation erörtert wurden. Vor allem der Punkt eines nachvollziehbaren Navigierens innerhalb der Graphendarstellung erhärtete sich.

Die Summe dieser Forderungen zeichnete schlussendlich SVG als klaren Favoriten ab. Der Standard ist offen und wird von einer starken Community stets weitergeführt. Das Format ist nicht wie bei den meisten Alternativen binär, sondern textueller Natur. Eine gewöhnliche XML-Instanz mit den Namensraum *svg,* wodurch auch die Frage der nachträglichen Bearbeitbarkeit geklärt ist. Gerade in diesem Bereich konnte sich keine einheitlich getragene kollaborative Lösung abzeichnen. Dieser Punkt wird noch an anderer Stelle nämlich in Bezug auf die Sicherheit durch Verschlüsselung noch näher erläutert werden.

8.3 Die Festlegung des Entwicklungssystems

Die wesentlichsten Faktoren in Bezug auf die Ein- und Ausgabeschnittstellen von RBMST wurden somit festgelegt. Der nächste Schritt gilt nun er Evaluation und Auswahl eines adäquaten Entwicklungssystems. Gemäß den impliziten Anforderungen nach Plattformunabhängigkeit sowie einer monolithisch vorwiegend im „Look & Feel“ stets gleichbleibenden GUI-Schnittstelle grenzt die Auswahl etwas ein. Da sich auch das Thema der Verarbeitungsgeschwindigkeit stellt, sollte das System auch einen einfachen Idiomatismus zur parallelen Verarbeitung zur Verfügung stellen.

Die zuvor vorgestellten Anforderungen, seien sie grafischer Natur, oder die Fähigkeit zur standardisierten Transformation und prägnante Parsingfähigkeiten setzen allesamt eine große Verfügbarkeit von stabilen und frei verwendbaren Bibliotheken voraus. Da wie schon zuvor erklärt, einer der ersten Implementierungsansätze auch die Verwirklichung einer eigenen internen DSL war, ergänzte auch diese Bedingung den Anforderungskatalog eines adäquaten Entwicklungssys-

[1] http://protege.stanford.edu/products.php

[2] eXtensible Stylesheet Language Transformation

tems. Nach mehrwöchigen Evaluationsversuchen engte sich Wahl zwischen Erlang & Scala ein.

Die Entscheidung war schwer, erstens hatte ich im Zuge meiner Entwicklerkarriere sehr positive Erfahrungen mit Erlang machen dürfen, und zweitens stellt Erlang mit der OTP[1]-Plattform ein Laufzeitsystem dar, das keine Wünsche bezüglich Bibliothekenvielfalt offen lässt. Auch das Aktorensystem, ähnlich dem asynchronen Messaging über Mailboxen, sowie das hot-code swapping ist den meisten seiner Konkurrenten stark überlegen [AJ2009]. Diese Vorteile waren aber für das Projekt RBMST jedoch nicht applikationsrelevant. Im Gegenteil, da die Sprache nicht so breit in der Community vertreten ist, sowie auch die Sprachperformance die sich eher im Mittelfeld bewegt, fiel letztendlich die knappe Entscheidung auf die zweite Auswahl.

8.3.1 Scala

Es ist ein Laufzeitsystem , das seit ca. 10 Jahren von der technischen Universität Lausanne (EPFL) unter der Führung von Dr. Martin Odersky stetig weiterentwickelt wird, und sich zunehmender Beliebtheit erfreut. Scala[2], wie der Name schon andeutet, skaliert von kleinen Skriptaufgaben bis hin zu verteilten Applikationen mit Komponenten-Charakter. Es wird auch eine sehr kompakte und vielseitig verwendbare Build-Umgebung namens *sbt*[3], das simple build tool, welche eigentlich eine interne DSL darstellt, angeboten [HD2013].

8.3.1.a Lizenzmodell

Die Lizenzgebarung von Scala ist sehr benutzerfreundlich, sie entspricht einer vereinfachten Form der BSD-Lizenz, ohne der Werbeklausel, die ansonsten bei jeder Produktpräsentation mitangeführt werden müsste. Im Konkreten handelt es sich hierbei um die sogenannte 3 -Klausel BSD Lizenz:

„Dieses Lizenzmodell unterscheidet sich von der GNU General Public License (GPL) darin, dass es kein Copyleft enthält. Ein Programmierer, der ein unter einer BSD-Lizenz veröffentlichtes Programm oder eine Bibliothek verändert und dann binär verbreitet, ist nicht verpflichtet den Quellcode mitzuveröffentlichen. Falls er es doch tut, dann muss das weiterhin unter BSD-Lizenz erfolgen. Wichtig ist vor allem zudem, dass er die BSD-Lizenz beim binären Veröffentlichen z. B. der Dokumentation hinzufügt und beim Quellcode direkt eingefügt belässt.“[4] Ergänzend dazu [SL2014].

Diese genaue Betrachtung des Lizenzrechtes ist im Hinblick auf die optionale Nutzung von bestehenden Source-Code Schablonen oder auch von Quellcode der mitgelieferten Bibliotheksmodule in Bezug auf RBMST entscheidend. Das zugrundeliegende Lizenzmodell kann grundsätzlich den weiteren Verlauf einer fremdgeführten Software-Entwicklung entweder forcieren oder auch nachhaltig hemmen.

---

[1] Open Telecom Platform

[2] http://www.scala-lang.org/

[3] http://www.scala-sbt.org/

[4]Zitiert aus http://de.wikipedia.org/wiki/BSD-Lizenz

Dieses Lizenzbestimmung ist somit aus Open Source Software wie auch aus kommerzieller Sicht geeignet.

8.3.1.b Systemcharakteristik

Die Spracharchitektur basiert prinzipiell auf der Java Virtual Machine Spezifikation (JVM), wobei es einst auch Portierungen für die .Net Plattform gab. Der Sprachkern kann dadurch mit dem klassischen Java-Compiler *javac* kompiliert werden, durch diese Interoperabilität ergeben sich viele Synergien. Einerseits kann von Java aus teilweise Scala Code direkt übernommen werden, andererseits kann aus Scala Perspektive jegliche Java Library transparent genutzt werden [WD2009].

Das Typsystem ist auf Java aufgebaut, und erweitert die Möglichkeiten erheblich. So sind beispielsweise Funktionen *first class member*, über das Schlüsselwort *object* können singelton Objekte erzeugt werden und der *Trait*-Mechanismus erweitert das klassische Java-Interface um default Felder und Methoden die schon während der Definitionsphase bereits implementiert werden können. Nicht nur jede Funktion beziehungsweise Methode liefert einen Rückgabewert, sondern auch nahezu jedes Idiom der Kontrollstruktur. Diese Forderung ist notwendig um das funktionale Sprachparadigma ebenfalls bedienen zu können. Die Sprachfunktionalität wird auf Bibliotheksebene abgebildet, und wirkt sich somit in Bezug auf die Optimierung der Applikationsgröße ebenfalls förderlich aus.

Der Sprachkern versucht mit konzisen Schlüsselwörtern und Grammatiken eine Fusion zwischen funktionaler wie auch objekt orientierter Idiomatik zu erreichen. Die Ausdrucksstärke dieser *objektfunktionalen* Sprache ist enorm, wobei eine steile Lernkurve vorauszusetzen ist. Die Java Technologie stellt den Wirt, man könnte somit zu Scala auch Java++ sagen.

Mittlerweile wird diese Technologie erfolgreich in etablierten Projekten wie *Twitter*[1] und *LinkedIn* eingesetzt. Die Toolchain ist bereits voll etabliert, Hersteller wie IntelliJ und Eclipse bieten schon seit geraumer Zeit IDE Umgebungen an.

Ein kommerzieller Support kann bei Bedarf über die Firma Typesafe[2] bezogen werden, die auch ebenfalls scalabasierte Plattformen und Frameworks, wie Akka und Play, für spezielle Einsatzzwecke entwickeln und kostenlos zur Verfügung stellen [GM2012].

Die Sprache derzeit in der Version 2.11.2, lässt sich aus den verschiedensten Bibliotheken zusammensetzten, es werden somit verschiedene Dialekte für den domänenspezifischen Gebrauch angeboten.

— scala-library, die Standardbibliothek deckt den regulären Sprachkern ab

— scala-compiler, das compiler front & backend

— scala-reflect, hierbei werden die Laufzeiteigenschaften der Typen repräsentiert

[1] https://twitter.com/
[2] https://typesafe.com/

— scala-xml, die gesamte Bibliothek zur XML Behandlung

— scala-parser-combinators, ein hoch modulares Parser-Generator Framework

— scala-continuations, Bibliothek für die spezielle cps-Programmiertechnik[1]

— scala-swing, ein aus der Java-Welt überführtes GUI-Framework

Das klassische Swing Framework kam auch bei den ersten Testversionen von RBMST zum Einsatz, wurde jedoch aufgrund seiner teilweisen Unzulänglichkeiten und dem bereits veralteten Programmiermodell zugunsten von ScalaFX verworfen.

Es bietet wie seine Stammimplementation JavaFX V2.2 [WJ2012], eine wesentlich konzisere Syntax und ein vereinheitlichtes *Event handling* Modell an. Dazu wurde eine interessante DSL entworfen um ein natürlicheres Binden an verschiedenste Objektparameter in kompakter Form, ähnlich einer Zuweisung, zu ermöglichen. Ein plakatives Beispiel in Abbildung 8.6 veranschaulicht die kompakte DSL Syntax. In natürlicher Sprache abgefasst bedeutet dies – setzte die Objektfarbe grün, wenn der Mauszeiger über dem Objekt ist, anderenfalls rot.

```
color <== when(hover) choose Color.GREEN otherwise Color.RED
```

Abbildung 8.6 – ScalaFX DSL zur kompakten Objektbindung

[1]continue passing style, wird häufig bei Bibliotheksentwicklungen verwendet

9 Kapitel

Die konzeptionelle Architektur des Software Prototypen

"Man versteht etwas nicht wirklich,
wenn man nicht versucht, es zu implementieren."
Donald Knuth

Schwerpunkte in diesem Kapitel:

- Der Software Prototyp als Forschungssoftware
- Die Auswahl eines adäquaten Vorgehensmodells
- Die Software-Architektur
- Aspekte der Performance und Ergonomie

9.1 Der Software Prototyp als Forschungssoftware

Die vorherigen Kapitel dienten der Findung geeigneter Implementierungsansätze. So wurde zuerst die Schnittstellenvielfalt des Wirtsystems diskutiert und anschließend im Bezug auf die gestellten Anforderungen vertieft. Durch die iterativen Vorgehensweisen die in kollaborativen Sitzungen stattfanden, wurde ein gemeinsames Verständnis der Mensch-Computer-Interaktion, kurz *HCI,* mit RBMST aufgebaut [HA2012].

Zur tatsächlichen Verwirklichung des angestrebten Werkzeuges wurden in zahlreichen Vorversuchsstadien der Implementierung mit verschiedensten Technologien experimentiert um eine geeignete Lösung zu entwickeln, die den technischen Herausforderungen genauso wie den ergonomischen Anforderungen genügt. Wie schon in Kapitel 8 formuliert, wurden dabei einige Hauptanforderungen an die Applikation identifiziert, die nun nochmals der Reihenfolge nach stichwortartig zusammengefaßt werden, um der späteren Diskussion der Software-Architektur besser folgen zu können.

— lose Systemkopplung

— minimale Konfiguration

— Portabilität bezüglich Betriebssystemplattform

— graphische Visualisierungsmöglichkeit

— ablaufprozessgesteuerte Eingabeschnittstellen

— integer verschlüsselter Dokumentenaustausch

— Skalierbarkeit bezüglich künftiger Performanceanforderungen

— Robustheit bei Fehleingaben

— Systemoffenheit bezüglich der Schlüsselwortpragmatiken

— Bereitstellung einer dimensionslosen Kostenmetrik

Diese insgesamt 10 Anforderungen verkörpern somit die Kernimplementierung der Applikation. Der letzte Punkt, die Kostenmetrik, wurde noch seitens der Industriepartner im Zuge der Umsetzungsphase nachträglich interveniert, da es als ein praktisches Instrument bezüglich der Projektplanerfüllung angesehen wird. Dieser Punkt, der ohne agiler Vorgehensweise, siehe dazu auch [SV2009], nicht so einfach umsetzbar gewesen wäre, wird nochmals explizit in der Artefaktbeschreibung unter 9.4.3 behandelt.

9.1.1 Implementationsabgrenzung

Wie anhand der zusammengefaßten Anforderungen ersichtlich wird, ist die Bandbreite der zu erfüllenden Implementationsaspekte erheblich. Ohne einer vernünftigen Beschränkung würde das Vorhaben einerseits den Fokus einer wissenschaftlichen Abhandlung verlieren und eher den Charakter eines klassischen Softwareprojektes einnehmen, andererseits würde ein zu rigides Verfolgen unternehmenspezifischer Anforderungen, auch den Anspruch auf die generelle Anwendbarkeit einer potenziell ähnlichen Problemklasse verlieren.

Die hier dokumentierte Software nimmt in diesem Sinne den Stellenwert einer Forschungssoftware ein. Dies impliziert, dass der Funktionsumfang die adressierten Anforderungen gänzlich abdeckt, jedoch nicht mit den Anspruch auf uneingeschränkte Fehlerlosigkeit und Allgemeingültigkeit. Vor allem die uneingeschränkte Anwendbarkeit auf sämtliche Repositorien des darunterliegenden Versionskontrollsystems ist nicht haltbar, da viele Plattformspezifika wie unterschiedliche Zeichensätze und Dateisystemeigenheiten zu divergent sind. Auch unter den stetigen Bemühungen der Testphasen, besitzt sie teilweise einen prototypischen Charakter und dient ausschließlich Forschungszwecken zum Verifizieren zuvor aufgestellter Postulate und Hypothesen. In diesem Kontext wird auch keine wie auch immer geartete Unterstützung (Support) angeboten, und dementsprechend ist die Software als ein „AS IS“ Produkt zu klassifizieren.

9.1.2 Lizenzfindung

Um die Basis für eine weitere Entwicklung, die auch unternehmenspezifisch weiterverfolgt werden kann, zu schaffen, stellt der Verfasser dieser Dissertation, der zugleich auch der originäre Ersteller der Software ist, das Werk unter dieselben Lizenzbedingungen wie das Entwicklungssystem Scala.

Es kommt somit die Fassung der „3-Klausel BSD Lizenz“ zur ausschließlichen Anwendung. Weitere Details zum freigestellten Bezug der Software RBMST werden in Kapitel 14.3.1 angeführt.

9.2 Die Auswahl eines adäquaten Vorgehensmodells

In Kapitel 1.3.2 wurde bereits auf die grundsätzliche Methodologie eingegangen, sie stellt somit das Framework, das Rahmenkonzept dar, konkretisiert jedoch nicht die Vorgehens- und Realisationsweisen im Detail, es muss somit ein Vorgehensmodell der Software-Entwicklung gewählt werden.

Das Projekt ist aus entwicklungstechnischer Sicht relativ komplex, jedoch in Ausstattung der Entwicklungskapazität sicherlich klein. Die Implementierung wird nicht verteilt ausgetragen, der Anforderungsanalyse hingegen wird eine besondere Aufmerksamkeit gewidmet. Die generellen Entwicklungskosten sind aus der Sicht der Probanden als gering und kalkulierbar zu bewerten, der zeitliche Horizont ist somit dehnbar. Aus Sicht der Implementation jedoch als unstetiges zeitliches Engagement zu bewerten, da auch neue Technologien zum Einsatz kommen.

Da das Produkt für keinen kommerziellen beziehungsweise Produktiveinsatz geplant ist, beschränkt sich die Testtiefe außerhalb der gewöhnlichen Entwicklungstests auf das Testumfeld der Probanden selbst. Die Anforderungen werden kollaborativ in mehreren Sitzungen erarbeitet. Fokus darauf, ist ein gemeinsames Verständigen auf die Schlüsselwortpragmatiken und die Definition der Benutzerschnittstellen. Dieser Prozess ist iterativer und inkrementeller Natur zugleich, und fluktuiert besonders während der Erfassungs- und prototypischen Testphase.

Aufgrund der zuvor beschriebenen Projektcharakteristik ergibt sich eine beschränkte Auswahl an klassischen wie auch agilen Vorgehensmodellen. Es konnte diesbezüglich eine phasengetriebene Entwicklung wie das Wasserfallmodell a priori ausgeschlossen werden. Da auch die notwendige Vorgehensweise wenig metamodellgetrieben erscheint, wurde ein Unified Process ebenfalls verworfen.

Am adäquatesten in Bezug auf klassische Vorgehensmodelle erschien eigentlich das Spiralmodell von Böhm welches auf anfänglich versucht wurde anzuwenden. Es stellte sich jedoch in Bezug auf eine nicht wirklich qualifizierbare und somit fehlende Risikoanalyse die Sinnhaftigkeit dieser Vorgehensweise in Frage. [BB2002]

9.2.1 Agile Vorgehensweise

Es konkretisierte sich durch die anfänglich labilen Anforderungen wie auch das prototypische Testen beim und durch den Kunden, dass ein agiles Modell zum Zug kommen sollte. Diese Vorgehensweisen entsprechen prinzipiell den klassischen Prozessen jedoch mit einigen signifikanten Unterschieden, wie zum Beispiel der Fokus auf Implementieren anstatt Spezifizieren und Dokumentieren, und somit die schnellere Ausrichtung auf noch zu konkretisierende Anforderungen.

Eine gut fügende agile Vorgehensweise wäre *Scrum* oder auch *eXtreme Programming*, welche gemäß folgendem Leitsatz kurz charakterisiert werden kann:

„a lightweight methology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements.“[BK2000]

Wobei agile Vorgehensweisen für Teams unter 3 Personen eher ungeeignet sind, können dennoch Konzepte wie verstärkte Kundeneinbindung und Entwicklungssprints übernommen werden [DF2008]. XP empfiehlt, im Gegensatz zu Scrum, Methoden wie „Pair Programming“, dennoch wird es nicht durch das Modell erzwungen – es eignet sich jede Methode die einer Qualitätsverbesserung dient. Einen entwicklungstechnischen Freiheitsgrad den Scrum gegenüber XP einräumt, ist das *nicht* strikt einzuhaltende Entwickeln gemäß einer Kundenpriorität. Ein agiles Vorgehen forciert somit die kollaborative Spezifikationsentwicklung.

9.3 Die Software-Architektur

Die Modellierung, Strukturierung und Dimensionierung zur Verwirklichung des ersonnenen Softwaremodells ist eine der wichtigsten Entwurfsentscheidungen die das Fundament der weiteren Implementation darstellt [RJ2006]. Es gibt für die verschiedensten Einsatzdomänen die vielfältigsten Baupläne, die allesamt abhängig vom Kernproblem in einer mehr oder weniger passenden Form angewandt werden können – eine perfekte Lösung, quasi ein Kochrezept, gibt es nicht. Man kann generell die Anforderungen in jeglicher Dimension betrachten, seien es die dynamischen Aspekte der Kommunikationsparadigmen, die zur Lastverteilung benötigte Prozessstruktur, oder auch die statischen Bedingungen die die Schichtenaufteilung auf Modulebene bestimmen.

Diese und jene Faktoren geben somit ein Gerüst vor, auf dem die Implementierung konsequent ausgeübt werden muss. Meistens sind es wie zuvor schon angedeutet, Gegebenheiten wie Schnittstellenspezifikationen und Anforderungen an die Rechenleistung die die Planung maßgeblich vorantreiben, aber auch orthogonale Aspekte wie jener der Plattformunabhängigkeit, einer zukünftigen Skalierbarkeit oder obligatorische Systemkonfigurationen die als Parameter in die Konzeption mit einbezogen werden müssen [EB2009]. Die konkrete Implementationsanforderung seitens RBMST endet mit einer möglichen Software-Architektur wo wir der Meinung sind, dass sie den Anforderungen gut entspricht und vor allem einer zukünftigen Weiterentwicklung alle Möglichkeiten offen lässt. Die nachfolgende Abbildung unter 9.1 soll die prinzipiellen Modulblöcke vorerst veranschaulichen.

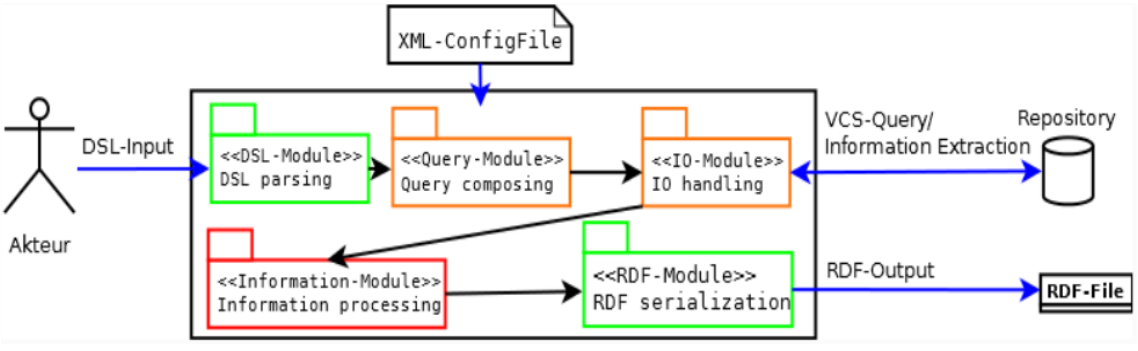


Abbildung 9.1 – Modulblöcke der Software RBMST

Das System besitzt mehrere Konfigurationsmechanismen, eine Kommandozeilenabarbeitung, eine zentrale Konfigurationsdatei auch konzipiert für den Stapelverarbeitungsmodus, sowie die GUI, eine ergonomisch geformte graphische Bedienschnittstelle.

Es sind in geblockter Symbolik einige Teilprozesse gekapselt dargestellt, wobei sich in Wirklichkeit die Funktionalität über diese Modulgrenze hinaus bewegt. Zum Beispiel im Fall der RDF-Inferenz, die so gesehen hier nicht explizit dargestellt ist, aber einen transienten Übergang zwischen Informationsverarbeitung und RDF Serialisation darstellt.

Es folgt eine plakative Beschreibung sämtlicher Module und möglicher Interaktionsformen wie zum Beispiel die zentrale Konfigurationsdatei. Sie stellt aus logischer Sicht den Anfang der weiteren Erklärungen dar.

9.3.1 Die zentrale XML-Konfigurationsdatei

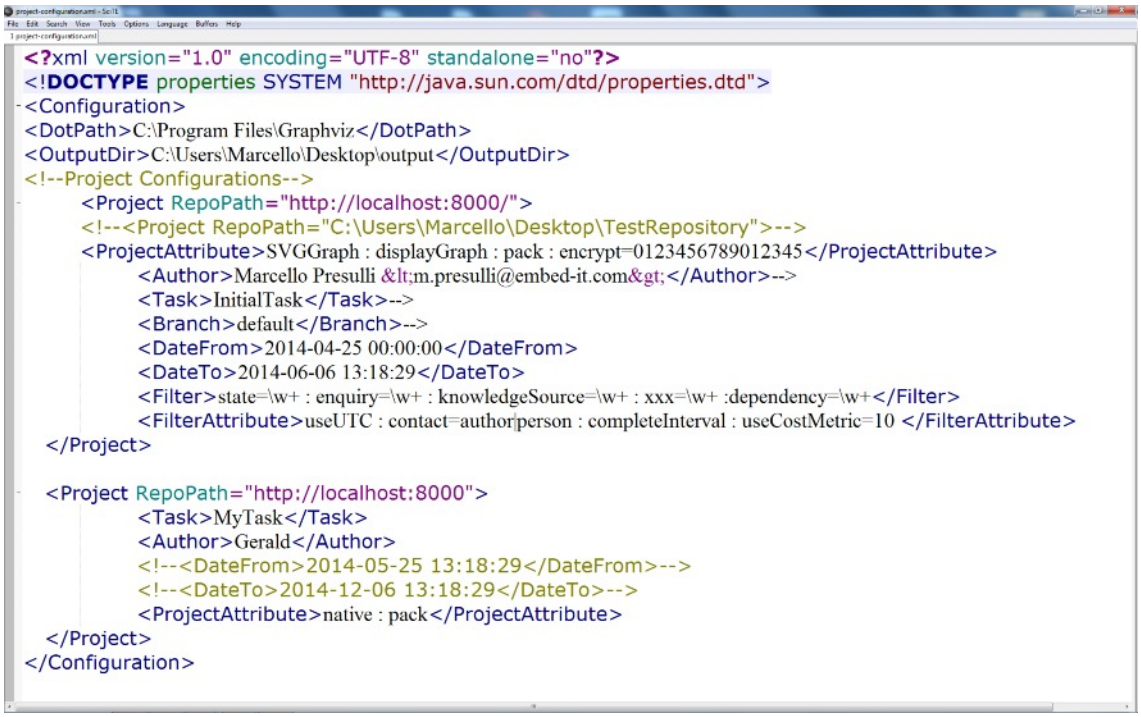
Die Initialisierung aller notwendigen Grundparameter einer Software während der startup-Phase gilt allgemein als sehr wichtig. Durch die Art und Weise entscheidet sich ob gewisse Einstellungen permanenter Natur sind oder auch ein dynamische Laufzeitänderung zulassen. Abgesehen von diesen Aspekten ist vor allem die Parameterübergabe in Bezug auf eine Betriebssystemabhängigkeit zu prüfen.

Werden etwa Parameter per Systemvariablen übergeben, hängt es davon ab, ob das konkrete Betriebssystem auch diese Variable *per default* definiert hat. Auch syntaktische Unterschiede wie eine Pfadangabe mit Leerzeichen oder auch der UNC-Separator, bei MS-Windows zum Beispiel der Backslash ‚\' anstatt des üblich Schrägstrichs ‚/', können von unterschiedlicher Art sein. Es werden auch andere Techniken angewandt, wie das dynamische Durchsuchen von Pfaden und zentralen betriebssystemabhängigen Steuerdateien während der Initialisierungsphase umso von Variablen unabhängiger zu werden. Diese Techniken bedürfen jedoch einer genaueren Abfrage des konkreten Betriebssystems manchmal sogar eine genauere Unterscheidung in Bezug auf die eingesetzte Version.

Eine probate Lösung, die sozusagen einen Mittelweg beschreitet, ist die applikationsspezifische Konfigurationsdatei. Hierbei können alle notwendigen Parameter syntaktisch, wenn nötig sogar in einer angepassten domänenspezifischen Notation, anführt werden. Es ist dann eine Frage der Anwendungsdefinition ob sich diese Datei an einer arbiträren oder bestimmten Stelle im Dateisystem befinden muss. Falls der Dateiaufbau keine spezielle Formatierung besitzen muss, jedoch auch eine bestimmte Komplexität aufweist, ist eine standardisierte Auszeichnungssprache wie XML sicherlich empfehlenswert. Es wird aber an dieser Stelle angemerkt, dass die Verwendung von Auszeichnungssprachen auch nur dann sinnvoll ist, falls der Umfang und die Struktur des Konfigurationssatzes auch den Einsatz dieser Syntax begünstigen. Bei Verwendung dieser Serialisation bietet sich sofort der Vorteil einer großen Bandbreite an Sprachbibliotheken und eigenständigen Werkzeugen zur Weiterbearbeitung an.

Zusammenfassend kann man durch die obigen Feststellungen ableiten, dass durch die zentrale Bereitstellung aller erforderlichen Applikationsparameter, in einer wie XML standardisierten Serialisationsform, eine kompakte und größtenteils betriebssystemautarke Lösung bereitgestellt werden kann. In geeigneter Anwendung von JVM basierenden Sprachen können zusätzlich über die File/URI Abstraktion auch Inkompatibilitäten wie das Problem des Pfadseparators elegant gelöst werden.

Diese Technik macht sich auch RBMST zu nutze, indem es so alle erforderlichen Parametersätze zentral verwaltet. Die nachfolgende Abbildung 9.2 stellt eine typische Konfigurationsdatei dar, die alle mandatorischen wie auch optionalen Parameter beinhaltet.



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<Configuration>
<DotPath>C:\Program Files\Graphviz</DotPath>
<OutputDir>C:\Users\Marcello\Desktop\output</OutputDir>
<!--Project Configurations-->
    <Project RepoPath="http://localhost:8000/">
    <!--<Project RepoPath="C:\Users\Marcello\Desktop\TestRepository">-->
    <ProjectAttribute>SVGGraph : displayGraph : pack : encrypt=0123456789012345</ProjectAttribute>
        <Author>Marcello Presulli &lt;m.presulli@embed-it.com&gt;</Author>-->
        <Task>InitialTask</Task>-->
        <Branch>default</Branch>-->
        <DateFrom>2014-04-25 00:00:00</DateFrom>
        <DateTo>2014-06-06 13:18:29</DateTo>
        <Filter>state=\w+ : enquiry=\w+ : knowledgeSource=\w+ : xxx=\w+ :dependency=\w+</Filter>
        <FilterAttribute>useUTC : contact=author|person : completeInterval : useCostMetric=10 </FilterAttribute>
  </Project>

  <Project RepoPath="http://localhost:8000">
        <Task>MyTask</Task>
        <Author>Gerald</Author>
        <!--<DateFrom>2014-05-25 13:18:29</DateFrom>-->
        <!--<DateTo>2014-12-06 13:18:29</DateTo>-->
        <ProjectAttribute>native : pack</ProjectAttribute>
  </Project>
</Configuration>
```

Abbildung 9.2 – eine vollständige RBMST-Konfigurationsdatei

Eine genauere Strukturanalyse zeigt, dass die gesamte Konfiguration in einem XML-Dokument mit UTF-8 Zeichensatzkodierung abgefasst und der syntaktische Aufbau mittels einer Java eigenen Document Type Definition (DTD) validiert ist. Der Dokumentaufbau ist mit einer <Configuration> Tag Klammer gekapselt. Innerhalb dieser Klammerung gibt es offensichtlich einen Vorspann, nach diesem wiederholt sich eine <Project> Struktur.

9.3.1.a Mandatorische Auszeichnung

Der Vorspann, der für alle nachstehende <Project> Projekte gilt, ist eine solche Notwendigkeit.

Der <DotPath> verlangt, wie der Name schon andeutet, eine gültige Pfadangabe zum Dot-Tool. Das Dot-Tool das später in diesem Kapitel noch näher vorgestellt

werden wird ist ein unerlässliches Werkzeug zur Generation des SVG Ausgabeartefakts.

Der <OutputPath> hingegen gibt einen geeigneten Pfad zur Persistierung aller möglichen Artefaktdateien an, er stellt somit den Ausgabepfad dar. Seine Notwendigkeit ist plausibel, und wird im Falle des nicht vorhandenen Verzeichniselements automatisch vom System generiert.

Implizit wird auch konkret die ausführbare Datei *hg* des Mercurial Versionskontrollsystems benötigt, diese muss jedoch nicht angegeben werden. Das System ermittelt über eine indirekte *hg* Versionsabfrage auch implizit dessen Existenz am Programmpfad ohne diesen dediziert parsen zu müssen.

Das letzte de facto Obligat bezüglich Konfiguration ist das Attribut RepoPath in jedem <Project> Tag. Über diese Eigenschaft wird der Pfad zum eigentlichen Repository, das von Interesse ist, bestimmt. Es bieten sich hierbei mehrere Adressierungsarten an.

Ein Pfad beginnend mit http:// signalisiert dem Benutzer, dass es sich dabei um einen fernen Zugriff handeln wird, jedoch auch nicht unbedingt muss. Denn diese Adressierung ist für die Webserverfunktionalität reserviert, wobei das ad hoc builtin Webservice mittels *hg serve* effektiv auch auf der lokalen Maschine instanziert werden kann. Die Syntax entspricht der normalen URL Adressierung samt Hostname, Portname und gegebenenfalls auch virtueller Verzeichnisname. Die Namenskonvention der Ausgabeartefakte setzt folgende Strategie ein. Falls ein virtueller Verzeichnisname vorhanden ist, also der letzte Teil des Pfades nach dem ‚/', dann wird dieser Name als Projektname verwendet, anderenfalls kommt der Hostname zur Verwendung.

Diese Syntaxform ist auch eng mit dem Web-Theming[1] von Mercurial verknüpft. Wie schon zuvor unter 8.1.2.c kurz angedeutet verwendet RBMST ein spezielles Template namens *rbmss* welches sich unter dem Tortoise Mercurial Pfad *templates* befinden muss. Die Anwendung erfolgt völlig transparent.

Es bedeutet, dass der Benutzer zwar den gewöhnlichen http-Repositorypfad angibt, der jedoch unter der Verwendung eines eigenen speziell für die Abfragen von *Logs* modifizierten Templates (rbmss) genutzt wird. Mittels dieser Technik können nun die angefragten Nutzdaten effizient *on the fly* in eine geeignetere Darstellung zur Weiterverarbeitung transformiert werden ohne den Benutzer direkt involvieren zu müssen. Zur Veranschaulichung der Syntax ein Auszug aus der Datei *logentry.tmpl* aus dem relevanten Verzeichnis rbmss.

```
changeset:   {node}
revision:    {rev}
user:        {author}
date:        {date}
summary:     {desc}
{branches%branchname}
```

[1] http://mercurial.selenic.com/wiki/Theming

{inbranch%branchname}
{tags%tagname}{bookmarks%bookmarkname}

Auflistung 9.1 – eine Template Datei von RBMSS

Wie hier in Auflistung 9.1 ersichtlich wird, hat die Syntax einen intuitiven Ansatz in Richtung Makroprogrammierung, da auch hier abstrakte Metakonstrukte zur Anwendung kommen um gewisse Repositoryinformationen wie Autor oder Revision zu erhalten.

Ein Pfad beginnend mit file:// oder auch einer normalen Pfadangabe, entspricht dem Operieren auf Dateisystemebene. Jedoch auch hier kann nicht wirklich unterschieden werden, es ist völlig transparent, ob es lokal oder über NFS eingebunden wurde.

Prinzipiell ist bei dieser Adressierungsform ein Pfadname immer vorhanden und somit folgt auch die konventionelle Benennung. Interessant dabei ist, dass wie in der obigen Konfiguration dargestellt, der Pfadname auch in der windowseigenen Backslash Notation erfolgen kann oder eben umgekehrt, beide Formen sind gleichwertig und können zuverlässig aufgelöst werden. Die klassischen Windows-Pfadangaben mit Whitespace (Blank) Zeichen stellen ebenfalls kein Problem dar, da jeglicher Pfad vor Verwendung in eine standardisierte URI gebracht wird.

9.3.1.b Optionale Auszeichnung

Zwischen der <Project> Klammerung werden weitere Tags angeführt. Diese haben allesamt die Aufgabe das Projekt näher im Sinne der Informationsextraktion und Artefaktausgabe zu spezifizieren. Diese Parameter sind jedoch, wie beim zweiten <Project> zu erkennen ist, optional. Das bedeutet ein Fehlen wird mit konventionellen Werten der Werkseinstellungen ersetzt. Bis auf das <ProjectAttribute> Tag, das die konkreten Ausprägungen der Ausgabeformen angibt, sind alle anderen fakultativer Natur.

Das <Author> Tag konkretisiert den Verfasser des Changesets wo das zu erfragende Filter manifestiert ist. Da der Lesezeichenmechanismus grundsätzlich asynchron zum Changeset ist, ist es immer der Autor der commit Operation und nicht ein etwaiger Fremdautor des Lesezeichens, im RBMSS Jargon, der Schlüsselwortpragmatik.

Das <Task> Tag bestimmt den zugeordneten Aufgabenbereich den es zu extrahieren gilt. Dieses Konstrukt wurde schon im Zuge der Schlüsselwortpragmatiken synonym als das Objekt im objekt orientierten Sprachgebrauch identifiziert. Ein Fehlen dieses Eintrags würde jedoch nicht wie in OOP einen syntaktischen Fehler auswerfen, sondern einfach alle prozessierten Schlüsselwortpragmatiken ergeben, jedoch ungefiltert in Bezug auf die Aufgabebereiche (Task). Andererseits ist es auch sinnvoll, die Aufgabenbereiche weiter mittels der Punktnotation zu unterteilen. Eine weitere denkbare Unterteilung der Tätigkeiten zum Beispiel in *Tätigkeit.Arbeitspaket.Aufgabenbereich1.dependency(Aufgabenbereich2)* könnte sinnvoll und nutzbringend sein um die Granularität weiter zu verfeinern. Aufgrund

der seiteneffektfreien Charakteristik der Lesezeichentechnik, stünde einem nachträglich akkordiertem Löschen der Verfeinerungen auch nichts im Wege.

9.3.2 Der Kommando Parser

Das Suchen nach Zeichenketten ist längstens zur einer soliden Technik etabliert, und wird in vielen Applikationen mittelbar oder unmittelbar eingesetzt. Es beginnt beim parsen von Argumenten der Kommandozeile bis hin zu subtilen Aufsuchen spezifischer Muster in einem großen Datenbestand, wobei sich die Suche nicht nur auf lesbare Zeichen beschränkt sondern auch auf binäre Konstrukte, manchmal auch als BLOB, ein Akronym für Binary Large Object, bezeichnet, anwenden lässt. Wie bereits an anderer Stelle schon erwähnt wird das grammatikalische Parsen von Tokenbeziehungen oft mit dem lexikalischen Scannen von Zeichenfolgen verwechselt.

Die Suchmuster entsprechen häufig regulären Ausdrücken, und werden oft in Form der EBNF[1] notiert. Das Laufzeitsystem Scala bietet über die Java Pattern, eine ausführliche stabile Bibliothek zur Mustererkennung an, wobei die Funktionalität von Parser-Generatoren durch eine Scala eigene Bibliothek namens parser-combinators vollständig abgedeckt wird, da diesbezüglich eine eigenständige DSL entwickelt wurde, die sich den verschiedensten Bedürfnissen sehr elegant anpassen lässt. Um nun die weiteren Möglichkeiten des RBMST-Parsers zu verstehen, wird in der folgenden Abbildung 9.3 die Syntax der häufigsten regulären Ausdrücke kurz dargestellt.

Characters

x	The character *x*
`\\`	The backslash character
`\0`*n*	The character with octal value `0`*n* (0 <= *n* <= 7)
`\0`*nn*	The character with octal value `0`*nn* (0 <= *n* <= 7)
`\0`*mnn*	The character with octal value `0`*mnn* (0 <= *m* <= 3, 0 <= *n* <= 7)
`\x`*hh*	The character with hexadecimal value `0x`*hh*
`\u`*hhhh*	The character with hexadecimal value `0x`*hhhh*
`\x{`*h...h*`}`	The character with hexadecimal value `0x`*h...h* (`Character.MIN_CODE_POINT` <= `0x`*h...h* <= `Character.MAX_CODE_POINT`)
`\t`	The tab character (`'\u0009'`)
`\n`	The newline (line feed) character (`'\u000A'`)
`\r`	The carriage-return character (`'\u000D'`)
`\f`	The form-feed character (`'\u000C'`)
`\a`	The alert (bell) character (`'\u0007'`)
`\e`	The escape character (`'\u001B'`)
`\c`*x*	The control character corresponding to *x*

Character classes

`[abc]`	a, b, or c (simple class)
`[^abc]`	Any character except a, b, or c (negation)
`[a-zA-Z]`	a through z or A through Z, inclusive (range)
`[a-d[m-p]]`	a through d, or m through p: `[a-dm-p]` (union)
`[a-z&&[def]]`	d, e, or f (intersection)
`[a-z&&[^bc]]`	a through z, except for b and c: `[ad-z]` (subtraction)
`[a-z&&[^m-p]]`	a through z, and not m through p: `[a-lq-z]`(subtraction)

Predefined character classes

`.`	Any character (may or may not match line terminators)
`\d`	A digit: `[0-9]`
`\D`	A non-digit: `[^0-9]`
`\s`	A whitespace character: `[ \t\n\x0B\f\r]`
`\S`	A non-whitespace character: `[^\s]`
`\w`	A word character: `[a-zA-Z_0-9]`
`\W`	A non-word character: `[^\w]`

Abbildung 9.3 – Auszug aus den in RBMST verwendbaren Java-Patterns

Die Liste würde sich noch lange weiterführen lassen, jedoch ist hier schon ein kanonischer Aufbau von einzelnen Zeichen bis hin zu Zeichenklassen erkennbar.

[1] Erweiterte Backus Naur Form

Die Möglichkeiten lassen sich von POSIX normierten Klassen über die verschiedensten Quantoren wie *, +, ? bis hin zu klassischen parserspezifischen Methoden wie *lookahead* und *lookbehind* ausweiten [FJ2009].

9.3.2.a Die spezifische Adaption der Schlüsselwortpragmatiken

Die Darstellung der zentralen Konfigurationsdatei im Kapitel 9.3.1 zeigt bereits einige solcher *Patterns* auf. Die angewandten Muster in den <Filter> Tags stellen somit das werkseitige Suchmuster pro Schlüsselwortargument dar. Falls ein Schlüsselwort nicht solch eine explizite Darstellung besitzt, wird das Muster *\w+* impliziert. Wie in obiger Abbildung ersichtlich, entspricht vereinfacht ausgedrückt das Muster *\w+* einem allgemeinen Wort, einen *Identifier*.

Der Ausdruck *dependency=\w+* legt beispielsweise folgende Beziehung fest. Erstens ist das Argument explizit, implizit wäre es ebenfalls \w+. Zweitens das Argument des Schlüsselwortes *dependency* wird nur dann als positives Suchergebnis detoliert, wenn es auch den Anforderungen der Zeichenklasse \w+ genügt. Drittens stellt es auch eine Anforderung an die unternehmenspezifische Konvention.

Task1.dependency(Task2)

Falls nämlich eine Beziehung zwischen dem dependency-Argument und dem Taskteil, der den ersten Objektteil vor den Punkt darstellt, durch eine Semantik einer anderen Schlüsselwortpragmatik definiert wird, dann ist auch die Notation des Taskelements eindeutig bestimmt. Im Beispiel darüber wäre somit die Notation von Task2 die maßgebende Formbestimmung für Task1.

Damit ist auch die Offenheit der Systematik eindeutig. Jedwedes reguläres Suchmuster das den Konventionen von Java Pattern[1] genügt, ist somit anwendbar. Wie der Praxiseinsatz auch zeigt, werden auch häufig literale Muster eingesetzt. Unternehmen einigen sich intern auf einen beispielsweise vordefinierten (hard codierten) Präfix und danach eine bestimmte endliche Folge von Zeichen.

So könnten Muster wie *Entwicklung\.A\w{2}[0-3]* zum Einsatz kommen, da sich so sehr genau bestimmte Anforderungen an die Informationsstruktur abbilden lassen können. Der obige Ausdruck entspräche somit, in natürlicher Sprache formuliert, einer Gruppe aus der Abteilung Entwicklung beginnend mit A und zwei weiteren Buchstaben oder Zahlen, zuletzt jedoch gefolgt von einer abschließenden Zahl zwischen 0 und 3.

9.3.3 Die Erstellung von RDF

Die Generierung eines adäquaten RDF-Graphen, was eine der Hauptaufgaben des Projektes charakterisiert, stellt bezüglich der grundsätzlichen Anforderungen seitens RBMSS keine harte Herausforderung. Die Beziehungen der RDF-Tripel zueinander sind eindeutig und geradlinig. Der Erstellung einer durchgängigen XML-Serialisation zwecks semi-automatisierter Weiterverarbeitung, wie sie anfäng-

[1]Siehe http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

lich konzipiert war, hätte eine rein klassische Implementationsanforderung verkörpert [BC2007].

Die Komplexität steigerte sich jedoch schlagartig durch die Forderung einer gut aufbereiteten graphischen Visualisierung unter der besonderen Berücksichtigung, dass die Darstellung auf ein Dokument begrenzt ist und der Graph so kompakt wie möglich dargestellt wird, um damit ein Navigieren auf das Notwendigste zu begrenzen. Um diese Anforderungen mittels einer kompakten Notation erfüllen zu können wurde ein API benutzt die eine besondere Abstraktionsebene zur Verfügung stellt.

9.3.3.a Das Modul SCARDF

Hinter dem Akronym verbirgt sich der Name *SCAla-RDF*. Diese Schnittstelle, die eigentlich mehr einem Framework gleicht stellt durch ihre spezielle DSL besondere Leistungsmerkmale zur Verfügung. Dieses Modul hat einen geringen Abhängigkeitsgraphen zum klassischen Vertreter der javabasierten RDF-Bibliotheken, dem Apache Jena Framework[1]. Da es noch sehr jung ist, ist der Reifegrad zwar erstaunlich gediegen jedoch ist die Binärkompatibilität zu Scala noch sehr labil, und somit muss die Bibliothek bei jeden Versionssprung neu gebildet werden. Diese Vorgehensweise ist jedoch vertretbar, da das Entwicklungssystem selbst sich im Kern immer weniger ändert und somit hält sich die Verzahnung in Grenzen.

Um die oben gestellten Anforderungen nach genereller Kompaktheit zu erfüllen verfährt der Algorithmus folgendermaßen. Nach der Aufbereitung der Menge der extrahierten Schlüsselwortpragmatiken, werden die Informationsentitäten in Aufgabenbereich, Schlüsselwort und Inhalt zerlegt, es entspricht somit einem Tripel der Form *Task-Keyword-Argument*.

Diese drei Formen bilden nun Teilelemente des RDF-Graphen. Sie werden semantisch verknüpft mit den obligatorischen Elementen des Repositories, wie beispielsweise Datum, Autorname und Zweigname. Eine kompakte Repräsentation der darzustellenden Relationen ist eine wesentliche Vorbedingung. Hierbei wird der Weg über eine typisierte jedoch nicht bezeichenbare (anonyme) Knotenform der RDF Notation verfolgt.

Exkurs RDF Blank Nodes

Als ein fixer Bestandteil der RDF-Spezifikation, unterscheidet sich diese Form von anderen vor allem in Bezug auf die Namensgebung der Ressourcen. Sie stellt die Klasse der anonymen aber typisierbaren Ressourcen dar, und wird oft in Form einer anonymen Mehrfachinstanzierung angewandt. Durch diese Zwitterstellung wird sie auch in anderen Konstellationen, wie zum Beispiel bei den Triple Store Datenbanken, ungern verwendet. Dieser Fremdartigkeit wurde auch im Zuge der RDF-Spezifizierung eine eigene Referenzierungsform eingeräumt, die *<rdf:nodeID>*.

Im Folgenden ein kurzer Auszug aus der RDF Syntax Spezifikation Version 2004:

[1] https://jena.apache.org/

„Blank nodes in the RDF graph are distinct but have no RDF URI reference identifier. It is sometimes required that the same graph blank node is referred to in the RDF/XML in multiple places, such as at the subject and object of several RDF triples.

In this case, a blank node identifier can be given to the blank node for identifying it in the document. Blank node identifiers in RDF/XML are scoped to the containing XML Information Set document information item. A blank node identifier is used on a node element to replace rdf:about="RDF URI reference" or on a property element to replace rdf:resource="RDF URI reference" with rdf:nodeID="blank node identifier" in both cases."[RS2004]

Eine weitere Referenz aus dem offiziellen RDF Tutorial des W3C Konsortiums verdeutlicht die Möglichkeiten zur Typisierung anonymer Knoten, in der Literatur oft auch als bNode bezeichnet.

„The fundamental problem is that using Jane's mailbox as a stand-in for Jane is not really accurate: Jane and her mailbox are not the same thing, and hence they should be identified differently. When Jane herself does not have a URI, a blank node provides a more accurate way of modeling this situation. Jane can be represented by a blank node, and that blank node used as the subject of a statement with exterms:mailbox as the property and the URIref mailto:jane@example.org as its value. The blank node could also be described with an rdf:type property having a value of exterms:Person (types are discussed in more detail in the following sections), an exterms:name property having a value of "Jane Smith", and any other descriptive information that might be useful, as shown in the following triples.“

```
_:jane  exterms:mailbox <mailto:jane@example.org>.
_:jane  rdf:type        exterms:Person.
_:jane  exterms:name    "Jane Smith".
_:jane  exterms:empID   "23748".
_:jane  exterms:age     "26".
```

Abbildung 9.4 – anonymer typisierter RDF Knoten aus dem RDF Tutorial

Abbildung 9.4 zeigt das klassische Beispiel eines anonymen RDF Knotens an. Diese Technik wird auch in der Implementierung von RBMST konsequent eingesetzt. Die gewöhnlichen Beziehungen wie das Repository Subjekt spezifiziert über ein Prädikat welches häufig aus dem Namensraum der Dublin Core Metaterms entlehnt wird ein zumeist literales Objekt. Ein klassisches Beispiel ist die Beziehung einer dedizierten Repository Revision zu einem Datumseintrag. Hierbei stellt die Revision das Subjekt dar, die Relation dcterms:date ist das Prädikat und „2014-05-25 11:00:00“ stellt beispielsweise das konkrete Objekt dazu dar.

Es gibt aber auch spezielle Beziehungen, und diese werden genau in der Form der typisierten anonymen Ressourcenbeziehung abgebildet. Hierbei spielt die namenlose Ressource typisiert auf einen Typ aus dem Namensbereich rbmss, zum Beispiel *state*, das Subjekt welches ohne Benennung der konkreten Ressourcenins-

tanz aber über eine eindeutige Relation, dcterms:type, zur Repository Revision steht.

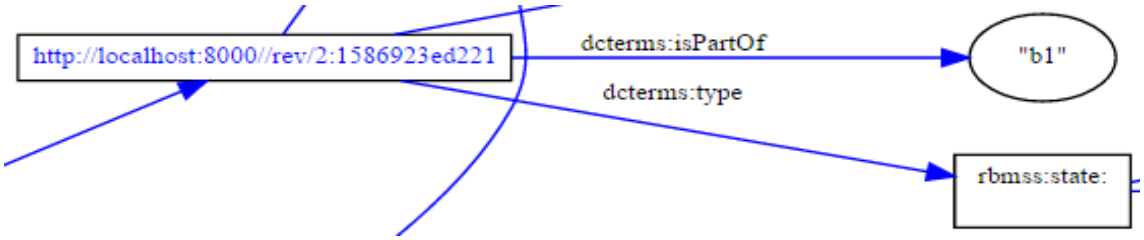


Abbildung 9.5 – Relation zwischen Repository-Subjekt und einem RBMSS-Objekt

Die Abbildung 9.5 verdeutlicht in diesem, zu einer größeren SVG-Darstellung zugehörigen, Abschnitt, die Beziehung zwischen dem Subjekt *Repository Revision* und dem Knoten rbmss:state über die Relation dcterms:type.

9.3.3.a.1 RDF Serialisierung

Nach erfolgter Graphenerstellung ist eine weitere Anforderung, dass jene Instanz auch in persistenter Form verfügbar gemacht wird. SCARDF bietet auch hierfür geeignete Mechanismen an, diese beziehen sich jedoch auf die Serialisation durch die Verwendung der einfacheren Turtle/N3 Notation. Zwecks einer erleichterten Austausch- und Bearbeitbarkeit über eine Semantic Web Toolchain, wird die XML-Notation jedoch strikt vorgezogen. Nicht zuletzt um auch eine konsistente Verfahrensweise mit der RBMST-Konfiguration über den XML-Mechanismus zu wahren.

Um eine tatsächliche RDF/XML-Serialisation zu erreichen benötigt man die direkte Interaktion mit dem zugrundeliegenden Jena Framework. Da sich diese Schnittstelle bereits in einem sehr ausgereiften Entwicklungsstadium befindet, bietet sie kompakte Methoden dazu an. Wichtig hierbei ist die Definition und Referenzierung der korrekten Namensräume, um auch die kompakte Notationssyntax in Form von *Namensraum:Typ* nutzen zu können, denn anderenfalls würde der URI in seiner gesamten Länge angeführt werden.

Insgesamt werden drei Namensräume maßgeblich verwendet, wobei nur einer, nämlich der Namensraum RBMSS, selbsterstellt und verwaltet wird sowie unter der URI-Referenz „http://purl.org/rbmss/1.0/#“ persistent zur Verfügung gestellt ist. Der Inhalt dieses Namenraums wurde bereits in Kapitel 6.4 vorgestellt. Die anderen beiden Domänen DCTERMS und VCARD werden von jeweils externen Konsortien verwaltet und gehostet.

9.3.3.b Die generische Reasoner Engine

Der Serialisierung schließt sich der Prozess der Inferenz an. Hierbei werden mittels eines speziellen Regelwerkes In- und Exklusionen in Anwendung diverser

Inferenzmechanismen vorgenommen und im Erfolgsfall auch transformiert [HJ2009]. Es werden somit Transformationsentscheidungen aufgrund der vorliegenden RDF/XML Serialisation abgeleitet und im positiven Fall *(true)* angewandt. Die Syntax der Regeln basiert auf der Jena Regel Maschine, sie stellt eine Instanz von drei verschiedenen Maschinen dar, den generischen, RDFS und OWL Reasoner. Ausschließlich die erste Form, im *forward chaining rule mode*, kommt bei RBMST zur Anwendung und nutzt die spezielle Syntax um einerseits unabhängig als Vorspann im *digraph*[1] und andererseits abhängig von Namensräumen auf Kanten und Knotenformen, anonym oder benannt, verschiedene Vortransformationen leisten zu können. Zum Beispiel dient die Regelklausel wie in Auflistung 9.2 der

```
//anonResource
(gv:graph gv:hasNode ?node)
isBNode(?node) -> (?node gv:shape "box")
(?node gv:fontsize "10px")
(?node gv:height "0")
(?node gv:width "0").
```

Auflistung 9.2 – eine Reasoner Regel für anonyme Knoten

Transformation eines *Bnode* (anonymer blank Node) hinsichtlich der graphischen Zuordnung einiger Primitive, wie hier am Beispiel gv:shape „box“. Wobei das ?node, die Knoten auf den die Operation stattfindet, eine initialisierte Variable der Rule Engine darstellt. Somit bekommt jede anonyme Ressource die gleichen graphischen Primitive übergeben.

Insgesamt stellt das Regelwerk eine Form der Prädikatenlogik mit Entscheidungscharakter dar. Deduktionen entsprechen prinzipiellen Funktionseinsprüngen mit Transformationsaufgaben. Teilweise sind einige Namensräume, wie RDF und RDFS seitens des Frameworks, bereits vordefiniert und müssen somit nicht eigens eingebunden werden. Weitere Informationen zu den einzelnen Regelformen finden sich in der Apache Jena Framework Documentation[2].

9.3.4 Die Generierung von SVG

Das letzte Glied der Artefaktkette, gilt der Erzeugung einer graphischen Darstellung. Es gibt eine Vielzahl von möglichen Formen, jedoch nur wenige die den graphischen Anforderungen von RBMST genügen. Die Hauptbedingungen wurden bereits an anderer Stelle formuliert, es ist die Forderung nach Navigationsmöglichkeit, Offenheit, Modifizierbarkeit und plattformunabhängige Darstellbarkeit womöglich durch Standardwerkzeuge.

Wenn man die oben aufgestellten Bedingungen in eine Schnittmenge mit den derzeit verfügbaren Grafikstandards bringt, bekommt man die Menge der möglichen Formate. Es beschränkt sich somit die Auswahl auf VML, SVG, EPS und

[1] Dot-Syntax die einen Graphenbeginn einleitet

[2] http://jena.apache.org/documentation/inference/#transitive

HTML-Canvas. Ersteres stellt den logischen, von W3C auch nicht ratifizierten, Vorgänger zu SVG dar, und ist daher überholt.

Diese restlichen Kandidaten wie EPS schieden einerseits aufgrund der erhöhten Komplexität der Syntax sowie der zu starken Ausrichtung auf Printmedien aus. Im anderen Fall von HTML5-Canvas ist ein Einsatz derzeit noch nicht zu empfehlen, da die Technologie zwar bereits als reif eingestuft werden kann, aber dennoch keine allzu breite Toolunterstützung erfährt. Überdies gibt es Vergleichsstudien, die die Verarbeitungsgeschwindigkeit des 2D-Rendering von SVG signifikant über HTML5-Canvas stellen [DJ2008]. Somit lässt sich SVG in Bezug seiner Auswahlkriterien folgendermassen beschreiben:

— eine Webtechnologie prädestiniert für die 2D-Browserdarstellung

— konsistente XML-basierende Syntax mit einfacher Struktur

— offener Standard, der von W3c gepflegt wird

— schneller Export auch für Binärgrafikformate möglich (SVG2PNG)

— Grafikformat ist rein textuell basiert

— Unabhängigkeit der Bildschirmauflösung (resolution independence)

— automatisches Prozessieren durch Javascript SVG DOM API möglich

Diese Punkte stellen die identifizierten Vorzüge in Bezug auf ihren Einsatzzweck dar. Es gibt sicherlich andere Anforderungsschwerpunkte, wie effizientes 3D-Rendering, komplexere Grafikprimitive oder eine optimierte Ausrichtung auf die Druckmedienerstellung, die den Einsatz von EPS, Canvas oder andere Formate präferenzieren würden.

9.3.4.a Die XSL Transformation

Durch die Festlegung des grafischen Ausgabeformates muss nun eine Transformation der ursprünglichen Datenbasis, die bereits in RDF/XML Serialisation vorliegt, angestoßen werden. Dieser Transformationsprozess hat die Aufgabe die Kanten und Knoten des RDF-Graphen in eine Zwischensprache zu übersetzen, die dann in automatisierter Form das Zielformat SVG erstellt. Dank der zugrundeliegenden XML-Struktur, ist diese Herausforderung mit einem klassischen Transformationsprozess bewerkstelligbar.

Bei dem angedachten XSLT Prozess, wobei *XSLT* eine *turing vollständige Sprache* ist und für eXtensible Stylesheet Language Transformation steht, stellt sich die Frage welche notwendigen Übersetzungen überhaupt formuliert werden müssen um letztendlich die Anforderungen der Zwischensprache zu erfüllen [SM2007]. Hierbei soll ein kleiner Codeausschnitt in Auflistung 9.3 zur Illustration der prinzipiellen Vorgehensweise dienen.

```
<xsl:for-each select="$it/gv:hasNode[not(@rdf:resource) and
 not(@rdf:nodeID) and @xml:lang='en']">
```

```
<xsl:variable name="isPreferredLanguage">
        <xsl:call-template name="isPreferredLanguage"/>
</xsl:variable>
<xsl:if test="$isPreferredLanguage='true'">
    <xsl:call-template name="eachLiteral">
        <xsl:with-param name="nodeValue" select="normalize-
        space(.)"/>
    </xsl:call-template>
</xsl:if>
</xsl:for-each>
```

Auflistung 9.3 – ein XSL-Ausschnitt der Datei rd2dot.xsl

Ein XSLT Prozess ist im Endeffekt eine implizite Verwendung der XPath Technologie um in einem wohlgeformten XML-Baum navigieren zu können. Dabei wird mittels Filterkriterien iterativ nach entsprechenden Tags und oder Attributen innerhalb der XML-Struktur gesucht. Im Erfolgsfall wird eine Schablone, ein sogenanntes Template, angewandt, dass je nach programmatischer Ausgestaltung verschiedene Variablenersetzungen beziehungsweise Textersätze in der neu generierten Transformation vornimmt.

Wie im obigen Codestück, ist das Suchen mittels des *„for-each select“* Idioms, gängige Praxis um nach gewissen Mustern innerhalb des Baumes zu suchen. Im Erfolgsfall, bei einem Match, und hier konkret zusätzlich durch den positiven Test des Konditionals *„if test“,* wird dann über das Konstrukt *„call-template name“* eine Schablone exekutiert, deren Rückgabewert direkt an dieser Stelle substituiert wird. Optionale Template Parameter, ähnlich einer Funktionsparameterübergabe, werden mittels *„with-param name“* übergeben. Es lässt sich auch unschwer erkennen, dass Variablen generell mit einem Dollar-Zeichen ($) gepräfixt werden.

Darunter kann man nun das Template, sprich die Quasifunktion eachLiteral, sehen, welches im obigen Ereignisfall zur Anwendung kommt. Der Kern dieser Funktion stellt den prinzipiellen Textersatz dar. Zuvor werden Variablen vereinbart die im späteren Textersatz verwendet werden.

Der Inhalt zwischen dem Tag <xsl:text> stellt somit den tatsächlichen Text der Ausgabetransformation dar. Es lässt sich auch die syntaktische Form des Escaping erkennen, hier wird durch das Literal *&quot* die Ausgabe *&quot* generiert, welche dann als ein weiteres Element in der Zwischensprache verarbeitet wird. Hier im konkreten Fall in Auflistung 9.4, findet der evaluierte Wert der Variable $value, umklammert von dem Literal *&quote,* als Textersatz Verwendung.

```
<xsl:template name="eachLiteral">
    <xsl:param name="nodeValue"/>
    <xsl:variable name="value">
        <xsl:call-template name="replace-string">
            <xsl:with-param name="text" select="$nodeValue"/>
            <xsl:with-param name="replace" select="'&quot;'"/>
```

```
            <xsl:with-param name="with" select="'&amp;quot;'"/>
        </xsl:call-template>
    </xsl:variable>
    <xsl:text>"&amp;quot;</xsl:text>
    <xsl:value-of select="$value"/>
    <xsl:text>&amp;quot;" [label="&amp;quot;</xsl:text>
    <xsl:value-of select="$value"/>
    <xsl:text>&amp;quot;",];</xsl:text>
</xsl:template>
```

Auflistung 9.4 – das exekutierte XSL-Makro zum vorigen Aufruf

9.3.4.b Der Dot Prozessor

Bei der zuvor diskutierten Zwischensprache handelt es sich um *Dot* [FJ2003]. Diese Sprache findet häufig in FOSS Tools ihre Verwendung, da sie bezüglich ihrer Lizenzbestimmung, vielfältige Verwendbarkeit und nicht zuletzt wegen ihrer Vielfältigkeit auch sehr beliebt ist. Ein Zitat soll in diesem Zusammenhang Dot kurz vorstellen.

„Graphviz ist ein von AT&T und den Bell-Labs entwickeltes plattformübergreifendes Open-Source-Programmpaket zur Visualisierung von Objekten und deren Beziehungen untereinander. Mathematisch ausgedrückt visualisiert Graphviz gerichtete und ungerichtete Graphen. Graphviz entnimmt alle zur Erzeugung der Grafik benötigten Anweisungen einer Textdatei, die eine Beschreibung der Knoten und Kanten des Graphen enthält. Die Positionen der einzelnen Knoten sowie die Krümmungen der Kanten werden aus dieser Beschreibung automatisch berechnet und dabei so optimiert, dass die Struktur des Graphen gut erkennbar ist. Zur Beschreibung des darzustellenden Graphen wird die Auszeichnungssprache DOT verwendet. Sie ist syntaktisch an die Programmiersprache C angelehnt. Graphviz bietet bei Bedarf auch zusätzliche Möglichkeiten zur Veränderung des Layouts sowie der Form und Farbgebung des Graphen. Oft genügt allein die Strukturdefinition des Graphen zur Erzeugung einer pasablen Ausgabe.

Daher können nicht nur Menschen, sondern auch automatische Prozesse Graphviz zur Erstellung von Visualisierungen nutzen. Auch können an vorhandenen Graphen sehr schnell Veränderungen vorgenommen werden, was mit einem Standard-Grafikprogramm nicht ohne weiteres möglich ist.“[1]

Da die Struktur der Ausgabe ein Graph ist, fällt damit die Wahl zur Einsatzbewertung von Dot nicht sonderlich schwer, mit dem Hintergrund dass es auch kaum eine ähnlich frei verfügbare Konkurrenztechnologie gibt. Dazu ein kurzer Codeabschnitt in Auslistung 9.5 um die graphenlastige Notationsform zu verdeutlichen.

```
digraph "Repository: localhost" {
    label="Created by mpr @ 2014-09-18 for http://localhost:8000/";
    center="true";
    nodesep="0.3";
```

[1] Quelle: http://de.wikipedia.org/wiki/Graphviz

```
    rankdir="LR";
    splines="true";
    overlap="false";
    ranksep="0.3";

    "A10" [ /*the node attributes */
    width="0",
    height="0",
    fontsize="10px",
    shape="box",
    label="rbmss:enquiry:\n"];
     "A10" -> "&quot;TaskMiddleware&quot;"[ fontsize="10px", color="blue",
    label="dcterms:isPartOf"];
}
```

Auflistung 9.5 – ein Graphenbeispiel in Dot-Syntax

Die Syntax ähnelt tatsächlich C, wie sich durch die Zuweisungsterminierung durch Semikolon (;) und die Kommentareinleitung (/*) und Blockstrukturierung durch geschwungene Klammern ({}) unschwer erkennen lässt [KS2010].

Das Literal „*A10*“ stellt hier beispielsweise eine Knotenbezeichnung dar. Die Pfeilsyntax (->) gibt eine Relation zwischen Knoten und Kante an. Die Ausdrücke in den eckigen Klammern, entweder hinter dem Knoten oder hinter der Kante, stellen jeweils Attribute zur graphischen Auszeichnung dar.

Im unteren Abschnitt wo die Konten-Kanten Relation dargestellt wird, kann man das wie zuvor diskutierte Literal *&quote* wiederfinden.Der gesamte Ausdruck steht somit für „*A10*“ als Subjekt das in Relation über die benannte Kante „dcterms:isPartOf“ zum Objekt „&quoteTaskMiddleware&quote“, also „TaskMiddleware“ steht.

Wie man hier gut erkennen kann, ist der syntaktische Aufbau in Anlehnung zur Graphennotation von verblüffender Intuitivität. Ein weiterer Vorteil von Graphviz Dot ist sicherlich die Form der Verfügbarkeit und die große Bandbreite an möglichen Ausgabeformaten, siehe dazu nachfolgende Abbildung. Durch die open source Gebarung steht das gesamte Package zum Download bereit, und stellt in diesem Fall auch eine komplette API zur Java-Anbindung zur Verfügung.

Die Community ist in diesem Bereich sehr stark, und bietet auch zu jeder veröffentlichten Version ein vorkompiliertes Programm für die gängigsten Plattformen an. Da die Grafikprimitive in schablonierter Form von RBMST generiert werden, sind die Anforderungen der Graphenvisualisierung bezüglich der verwendeten Dot-Syntax sehr gering.

Es wird dadurch das bereitgestellte Programm Dot seitens RBMST so eingebunden, dass es als Unterprozess durch den betriebssystemspezifischen Kommandointerpreter parametriert aufgerufen und exekutiert wird. Nachstehende Abbildung

9.6 führt sämtliche Transformationsausgabeformate des Dot-Prozessors in alphabetischer Reihenfolge an.

bmp	Windows Bitmap Format
canon dot gv xdot xdot1.2 xdot1.4	DOT
cgimage	CGImage bitmap format
cmap	Client-side imagemap (deprecated)
eps	Encapsulated PostScript
exr	OpenEXR
fig	FIG
gd gd2	GD/GD2 formats
gif	GIF
gtk	GTK canvas
ico	Icon Image File Format
imap cmapx	Server-side and client-side imagemaps
imap_np cmapx_np	Server-side and client-side imagemaps
ismap	Server-side imagemap (deprecated)
jp2	JPEG 2000
jpg jpeg jpe	JPEG
pct pict	PICT
pdf	Portable Document Format (PDF)
pic	Kernighan's PIC graphics language
plain plain-ext	Simple text format
png	Portable Network Graphics format
pov	POV-Ray markup language (prototype)
ps	PostScript
ps2	PostScript for PDF
psd	PSD
sgi	SGI
svg svgz	Scalable Vector Graphics
tga	Truevision TGA
tif tiff	TIFF (Tag Image File Format)
tk	TK graphics
vml vmlz	Vector Markup Language (VML)
vrml	VRML
wbmp	Wireless BitMap format
webp	Image format for the Web
xlib x11	Xlib canvas

Abbildung 9.6 – sämtliche Ausgabetransformationen von Graphviz Dot

9.3.5 Authentifizierung durch Kryptographie

Eine Forderung, die sich im Kontext der nachträglich gewünschten Bearbeitbarkeit des Ausgabeartefaktes ergab, war das diametrale Verlangen nach Sicherheit. Dieser Aspekt nimmt eine orthogonale Stellung bezüglich des prinzipiellen Hauptanwendungszweckes, der Erstellung einer Visualisierung einer teilöffentlichen Informationsbasis des Repositories, ein.

Nichts desto trotz sind die Intentionen der Beteiligten offensichtlich heterogen. Es treffen hierbei zwei Interessengruppen, salopp formuliert, die der Manager und der Techniker zusammen. Beide „Welten“ setzen andere Betrachtungsweisen und Maxime in ihrem Handeln. Konsensual ergab sich das Verlangen nach einer einfach zu praktizierenden Sicherheitsoption.

Um eine adäquate Lösung zu finden, müssen die Intentionen und Bedenken der Beteiligten im Vorfeld hinterfragt werden [SB1996]. Es konnten innerhalb der Ablaufstruktur getriebenen Projektkommunikationshierarchie folgende Unsicherheiten vage identifiziert werden, die hier in absteigender Priorität aufgelistet werden:

— unerlaubte Weitergabe eines effektiven Ausgabeartefakts

— nicht kollaborativ entschiedene nachträgliche Änderungen der Ausgabeartefakte

— Sicherstellung durch quasi Authentifikation des Senders und Empfängers

— Verbindlichkeitscharakter in Bezug auf den Ersteller bzw. Erstellungsdatum

Der erste Punkt nimmt hier definitiv den größten Stellenwert ein. Es gibt begründete Unsicherheit, dass zufällig oder auch bewusst Projektdarstellungen, intern auch oft bezeichnet als *Projektsnapshots*[1], die qualifizierte Audienz verlassen und somit ungewollt abteilungs- oder noch schlimmer unternehmensübergreifend verbreitet werden und somit Unbefugten zur freien Disposition stehen.

Auch der Fall einer Weiterleitung innerhalb der Kommunikationsstruktur, jedoch mit eigenwilliger nachträglicher Änderung der Information, wäre prekär. Es kann sich hierbei um bewusste Vortäuschung falscher Tatsachen handeln oder durch eine Andersdarstellung eine Entscheidung beeinflussen, wo jenes Artefakt als Entscheidungshilfe herangezogen wurde.

In diesem Kontext ist auch der integre Transport zwischen des Senders und des Empfängers unerlässlich. Es wäre fatal, wenn es keine Möglichkeit gebe einen gedachten Empfänger dediziert zu adressieren, in dem Sinn zu garantieren, dass nur er derjenige sein kann der auch die Nachricht interpretieren kann, der umgekehrte Fall der Integrität ist jedoch genauso wichtig.

Zu guter Letzt sollte auch ein einfacher Signaturmechanismus in Anlehnung an ein digitales Wasserzeichen [DJ2000] vorgesehen werden, der zumindest den Ersteller, das Erstelldatum und die genaue Referenz der Informationsquelle zum Zeitpunkt der Extraktion darstellt, ohne auf weitere digitale forensische Mittel wie die Metadaten des Autors oder das Erstellungsdatum der Datei zurückgreifen zu müssen, was letztendlich wieder ein erhöhtes Risiko an Unsicherheit darstellen würde.

Wichtig zu unterscheiden ist jedoch die Höhe beziehungsweise die Granularität der Sicherheit. Es ist generell innerhalb eines Unternehmen anzunehmen, dass sich gewisse Vertraulichkeitsbeziehungen während des Berufslebens etablieren. Somit wäre ein strenges Sicherheitskorsett zu rigide. Es soll nur im Bedarfsfall anzuwenden

[1] Damit ist ein wahrer Ausschnitt eines bestimmten Projektzustandes gemeint

sein und dann auch in einfacher Art und Weise, wenn möglich gar transparent. Auch die Höhe der Sicherheit als Ganzes betrachtet sollte in diesem Kontext auch nicht überbewertet werden, da es ansonsten den Kommunikationsverlauf eines Projektgefüges signifikant verkomplizieren könnte.

9.3.5.a Die symmetrische Verschlüsselung

Die grundsätzlichen Unsicherheiten verdichteten sich somit zu Anforderungen und ergaben so zu sagen als größten gemeinsamen Nenner, die symmetrische Verschlüsselung. Diese Form der Kryptographie besitzt einige inhärent positive Eigenschaften, die hier besonders gut zur Geltung kommen können.

So ist grundsätzlich, unter Voraussetzung eines tauglichen Kryptographiealgorithmus, eine hohe Sicherheit bezüglich dechiffrierens gegeben. Das bedeutet, dass im Fall einer ungewollten Veröffentlichung eines verschlüsselten Dokumentes ein unerlaubtes Lesen fast unmöglich ist. Durch die Symmetrieeigenschaft des gemeinsamen Schlüsselwissens, ergibt sich eine starke Sender -Empfängerbindung. Vorausgesetzt der Schlüsselaustausch verläuft verbindlich unter vier Augen, kann nur der Sender den einzig adressierten Empfänger den Schlüssel offenbaren, da ein Dritter theoretisch das dazu benötigte Schlüsselwissen nicht besitzen kann. Des Weiteren kann durch die Symmetrie auf die öffentliche Schlüsselverwaltung, wie sie in der PKI[1] vorgesehen ist, verzichtet werden, was wiederum der Anwendungseinfachheit entgegenkommt, trotz allem aber eine de facto Authentifikation garantiert [YL2011].

Eine nachträgliche Änderung der Artefaktausgabe wird ebenfalls erschwert, da abhängig von der konkreten Implementation, eine Prüfsumme (Hashwert) meist in Form von *MD5* oder SHA-1 während der Verschlüsselung implizit mitverwendet werden kann um die Integrität der zu verschlüsselnden Daten zu gewähren. Diese Form der Integrität wird von RBMST auf einer einfacheren Ebene, die den praktischen firmeninternen Einsatz genügt, realisiert, nämlich in Form einer einfachen Signatur, ähnlich einem Wasserzeichen, während der Artefakterstellung.

Diese kryptographischen Anforderungen werden mit dem Java Krypto-Subsystem gänzlich abgedeckt. RBMST verwendet in diesem Zusammenhang den AES Verschlüsselungsmechanismus mit der Standardschlüssellänge von 128 Bit im einfachem ECB[2] Verfahren, wobei dementsprechend der Schlüssel mindestens 16 Stellen lang sein muss.

Mit der zusätzlichen Möglichkeit während der Artefakterstellung auch die GZip Kompression zu aktivieren um beispielsweise die Größe der Ausgabedatei zu komprimieren ergibt eine generierte Prüfsumme (CRC-32) zusätzliche Sicherheit in der Datenintegrität [HE2006]. Im Folgenden wird kurz in Auflistung 9.6 auf die kompakte idiomatische Anwendbarkeit der Krypto-Bibliothek aus Sicht der RBMST-Implementation eingegangen.

val cipher = getInstance("AES/ECB/PKCS5Padding")

[1] PKI: Public Key Infrastructure

[2] ECB: Electronic Code Book

```
val ProjectName=readConfiguration.config("ProjectName")
val outputPath=new File(readConfiguration.config("OutputPath"))

new java.util.zip.GZIPOutputStream(
new javax.crypto.CipherOutputStream(new FileOutputStream(
new File(outputPath, ProjectName+".gz.aes")
), cipher))
```

Auflistung 9.6 – die Datenstromverkettung von GZip und Cipher

Der erste Codeabschnitt vereinbart die benötigten Variablen wie Verschlüsselungsmodus, Projektname und Ausgabepfad die im unteren Abschnitt verwendet werden. Im zweiten Abschnitt sieht man, durch die lange Verkettung von Datenströme, dass die Daten in diesem Betriebsmodus vorerst mittels *GZip* komprimiert und danach mit *AES* verschlüsselt und im Ausgabepfad als Artefakt unter dem jeweiligen Projektnamen abgelegt werden.

9.4 Aspekte der Performance und Ergonomie

Zur Darstellung der Software Architektur gehören nicht nur die Erläuterungen und konzeptionellen Erörterungen der einzelnen Modulblöcke die in der ersten Analyse einer Software nicht offensichtlich sind, sondern auch Gesichtspunkte die vom technischen Verständnis her sekundären Charakter haben, aus der Sicht der Benutzer jedoch essentiell sind. Denn meist sind es Faktoren wie zu lange Wartezeiten, nervenaufreibende Eingabeoberflächen oder ungenügend strukturierte Repräsentationen die eine längere Benutzung subjektiv unzumutbar werden lassen.

9.4.1 Die logische Sicht des Datenflusses

Die statische Architektur der Software ist für eine stabile Entwicklungsbasis unabdingbar. Genauso wichtig sind jedoch auch die dynamischen Details, wie die Benutzungsfrequenz oder beispielsweise der wiederkehrende Aufruf diverser Konvertierungsfunktionen. Die Summe dieser dynamischen Aufrufverkettungen kann, abgesehen von den zu bewältigenden Daten die sowieso ein Faktum für sich darstellen, zu erheblichen Bearbeitungszeitverzögerungen kommen. In diesem und ähnlichen Zusammenhängen wird bei suboptimaler Behandlung auch oft von *waste of CPU time*[1] gesprochen.

Die logische Sicht auf einen effizienten Datenfluss innerhalb der Software-Architektur ist hier sicherlich ein probates Mittel um ein ausgewogenes Verhältnis der involvierten Prozesse zu erlangen. RBMST adressiert diese Sichtweise, ähnlich einem Datenfluss-Diagramm [BR2002], in dem es eine Top-Down Betrachtung der notwendigen Informationsbeziehungen ins Kalkül zieht.

In diesem Zusammenhang kann auch hier eine Analogie zur *PushDown* Technik die bei relationalen Datenbanken, oft zur Minderung der Anzahl redundanter

[1] Vergeudung der Prozessorzeit

Datensätze Verwendung findet, gezogen werden. Im Wesentlichen geht es darum sich über die involvierten Formen und deren N-ären Beziehungen im Klaren zu werden.

Beispielsweise müssen nicht alle Changesets eines Repositories bedingungslos erhoben werden um danach sukzessive die Lesezeichen, sprich aus Informationssicht die Schlüsselwortpragmatiken, zu extrahieren oder alle Datensätze sequenziell einer Bedingung nach der anderen zu filtern. Wie schon oben erwähnt ähnelt diese Datenselektion dem *Pushdown* Verfahren, wo mittels der relationalen mengenbasierten Sprache SQL diejenigen Tabellen erfragt und Schnittmengen gebildet werden, die in weiterer Folge die Basis für weitere Selektionen stellen.

Als Beispiel soll in Auflistung 9.7 ein RBMST Codeabschnitt aus dem Bereich der Repositoryabfrage via HTTP dienen.

```
val bookmarks=
scala.io.Source.fromURL(repo+"/"+theme+"-bookmarks/").mkString

(filterSet.par.map { filter =>
 val (bmName,bmArg)=splitBookmarks(filter)
 val pattern=
(task+"\\."+bmName+"""\("""+bmArg+"""\)R\d+\s+(\p{XDigit}{40})""").r("id")
 filter -> (bookmarks.split("\r?\n") collect { entry =>
   (pattern findFirstMatchIn entry) match {
    case Some(bm) =>
    val blockslice=scala.io.Source.fromURL(repo+"/"+theme+"-log/"+
   bm.group("id")).getLines.toSeq.
   dropWhile(!_.contains("changeset:")).takeWhile(_.nonEmpty)
```

Auflistung 9.7 – eine effiziente Datenstromselektion

Die *bookmarks* Anweisung ermittelt alle Lesezeichen eines HTTP-Repositories. Danach wird der relevante Datensatz (bmName, bmArg) aufgesplittet, gemäß einem vorgegebenen Muster analysiert und das dazugehörige Changeset mittels der *blockslice* Anweisung ermittelt, und bis zu einem bestimmten *Terminator* (Trennsymbol) eingelesen. Mittels dieser Filtertechnik werden auch nur die tatsächlich benötigten Daten behandelt, und der Prozess belastet das Gesamtsystem nur in der notwendigen Weise.

9.4.2 Parallelismus als inhärenter Faktor der Datenstruktur

Das Entwicklungssystem Scala stellt auch zur Steigerung der Performance eine sehr interessante Idiomatik zur Verfügung. Sie besteht bei Verwendung der Scala eigenen Collections[1], die Standardbibliothek des Laufzeitsystems mit gängigen Datenstrukturen wie Array, Vector und List und funktionaler Ausrichtung hinsichtlich klassicher Operationen wie foreach, map, fold und collect.

[1]Benannt nach Kollektionen, einer Aneinanderreihung von Datenblöcken

Ein spezielle Untergruppe dieser Bibliothek widmet sich der Modellierung paralleler Datenstrukturen und bietet dazu elegante Methoden, für die parallele Behandlung großer Datenmengen, an [SJ2012]. Die Hardwareausstattungen heutzutage schaffen zwar die Möglichkeiten dafür, aber nur wenig direkte Unterstützung findet sich in den verfügbaren *Mainstream* Programmiersprachen. Es ist oft schwierig korrekt parallel arbeitende, in der Literatur oft auch als konkurrierende, Programme zu entwickeln, wobei es Anforderungen oft nahe legen oder sogar aufgrund ihrer natürlichen Struktur verlangen.

```
scala> (1 to 10).par foreach println
1
4
3
7
2
9
6
8
10
5
```

Abbildung 9.7 – die par Methode und ihre Verwendung

Wie in Abbildung 9.7 dargestellt, implementiert die *par* Methode das parallele Abarbeiten von Operationen auf Kollektionen, wenn immer es implizit möglich ist. So zeigt eine unter der REPL ausgeführte Anweisung, eine ungeordnete Ausgabe der Zahlenfolge 1 bis 10. Ungeordnet deswegen, da die Folge offensichtlich parallel abgearbeitet wurde. Dies geschieht auf Basis der sequenziellen Collections durch implizites Aufspannen und Abarbeitung mehrerer Threads und ist somit sehr effizient.

In einer *for (i <- (0 until 100).par) yield i + " "* Schleife hingegen wo die Ergebnisse durch das *yield* Schlüsselwort in geordneter Form retourniert werden, wird die Sequenz natürlicherweise auch parallel abgearbeitet aber eben in konsekutiver Form zurückgegeben. Vorsicht ist geboten wenn ein geteilter Zustand parallel aktualisiert wird.

Wie in unterer Auflistung 9.8 verdeutlicht, ist hier der Wert der Variable count indeterministisch.

```
var count = 0
for (c <- (1 to 100).par) { if (c % 2 == 0) count += 1 }
println(count)      //Achtung: letzter Wert unbestimmbar!
```

Auflistung 9.8 – die Unvorhersagbarkeit paralleler Abarbeitung

Die parallelen Kollektionen entsprechen den Untertypen von *ParIterable* und sind somit nicht *kovariant* zu den sequenziell iterierbaren Typ *Iterable*. Sie können daher nicht als normales iterierbares Argument übergeben werden, und müssen gegebenenfalls vor Verwendung mittels der *ser* Methode konvertiert werden [LM2012b].

Diese Form der Datenstrukturbehandlung ist bei RBMST stark in Verwendung. Ein Codestück aus dem Bereich des Metadaten Parsings soll den Einsatz kurz durch Auflistung 9.9 veranschaulichen.

```
val sequence=data.par.collect {
case(key,value) if value.nonEmpty =>
val (bmName,bmArg)=splitBookmarks(key)

val  bookmarkFilter= ("bookmark:("+task+")\\."
    +bmName+"""\(("""+bmArg+""")\)R(\d+)""").r("task","arg","id")

val collection= value.par.flatMap {
case general(rev, node, date, author, branch, tagsBookmarks) =>
```

Auflistung 9.9 – parallel genutzte Datenstrukturen in RBMST

Die Verwendung der beiden *par* Methodenaufrufe im Kontext der *data* und *value* Kollektioneninstanzen erlauben ein paralleles Abarbeiten, hier im Konkreten das Parsen der vorher extrahierten Schlüsselwort-Teilmengen.

9.4.3 Die dimensionslose Kostenmetrik

Ein benötigtes Leistungsmerkmal das sich erst am Ende der Implementierungsphase ergeben hat, war der unternehmerische Wunsch nach einer hoch flexiblen Bewertungsmetrik. Diese Metrik soll deswegen flexibel sein da es sehr projekt- sogar phasenspezifisch ist, welcher Faktor für welche Bemessung zur Anwendung kommt. Es können empirisch ermittelte Erfahrungswerte wie auch betriebswirtschaftliche Faktoren wahlweise zum Einsatz kommen um einen projektierten Projekterfolg zu messen [FN2014]. Die frei parametrierbare Bewertung benutzt lediglich die verstrichene Zeit des Projektsnapshots gemessen zwischen beliebigen state Schlüsselwörtern die in einer gültigen Intervallfolge von *start-stop* Argumentatomen, wie bereits in Kapitel 7.5.4 erläutert, abgeschlossen sind. Gültige Folgen sind immer ein *start* gefolgt von einem *stop*. RBMST bietet hier auch die Möglichkeit der Kalkulation auf das volle Zeitintervall. Das bedeutet, falls der *Projektsnapshot* eine zeitlich dimensionierte Restriktion mittels der *dateFrom* und *dateTo* Tags aus der Konfigurationsdatei erfahren hat, werden die Beginn- und Endzeiten der Intervalle miteinkalkuliert.

Versehentliche Doppel *start* beziehungsweise Doppel stop wie auch unregelmässige stop-start Folgen werden dabei bewusst überlesen. Zur Verdeutlichung eine schematische Abfolge von start-stop Atomen. Das Schema setzt eine implizite Verwendung vom Schlüsselwort *state* voraus. Hier ein paar beispielhafte Folgen:

start - stop - start - stop	summiert die Zeiten zwischen *start - stop*
start - start - start - stop - stop	summiert die Zeiten zwischen dem *letzten start* und *erstem stop*
stop - start	überspringt *erstes stop* und beginnt bei *erstem start*

dateFrom - start - stop - dateTo zählt im Fall einer vollen Intervallbemessung zur obligatorischen *start - stop* Zeit auch die Zeitdifferenz zwischen dateFrom - *start* und *stop* - dateTo hinzu

Das resultierende Zeitintervall wird in Sekunden bemessen. Es ist also möglich mittels beliebiger Parametrierung auf Sekundenebene eine lineare Fortrechnung von Parameter, sprich Kostenfaktor, mal Sekunden zu erhalten. Ob der Faktor nun einer Geldeinheit entspricht, oder ideellen Werten, ist dabei irrelevant. Ferner ist festzuhalten, dass durch die zeitbasierte Betrachtungsweise und Variabilität der Bewertungsfaktoren eine Art der *dimensionslosen* Kostenermittlung vorliegt, wobei wohlgemerkt die Kosten auch einen negativen Wert darstellen können, und gegebenenfalls im betriebswirtschaftlichem Sinne einen eigentlichen Ertrag darstellen. Die Darstellung dieser Kostenmetrik wird durch ihre Granularität bedingt in doppelter Genauigkeit, somit als Gleitkommazahl vom Typ *double* gerechnet. Eine SVG-Teildarstellung in Abbildung 9.7 stellt die dimensionslose Ausgabe im Nachfolgenden plakativ dar.

Es sollte somit ersichtlich sein, dass die hohe Auflösung des Datentyps das Resultat bewusst *ungerundet* dargestellt. Die genaue Darstellung ist deswegen sinnvoll, da das System den Einsatzzweck der Kennzahl nicht beschränken möchte, und somit auch nicht die dazu fallweise nötige Interpretationstiefe.

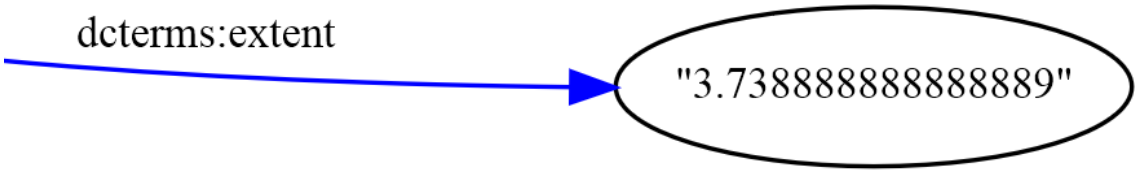


Abbildung 9.8 – die dimensionslose Kostenmetrik

9.4.4 Navigationsmöglichkeiten in der SVG Darstellung

Es wurde schon eingangs das Motiv zur Benutzung des SVG-Formates vorgestellt. Eine der Hauptforderungen an die Visualisierung stellte die plattformunabängige Darstellungstechnik mit ihren Navigationsmöglichkeiten, da einerseits abhängig von der Informationsdichte des Projektsnapshots und andererseits von der Größe und maximalen Auflösung des Visualisierungsgerätes die Darstellung mitunter große Ausmaße annehmen kann [MN2004].

Gerade diese Unzulänglichkeiten können mittels einer skalierbaren Vektordarstellung gut adressiert werden. In Kombination mit einfachen Werzeugen der Darstellung, wie die vertraute Browsertechnik, steht einem komfortablen Navigieren somit nichts mehr im Wege. Wie in der nachfolgenden Abbildung 9.8 ersichtlich, wird ein Hyperlink zur direkten Anbindung an ein HTTP-Repository zwecks vereinfachter Navigation angeboten.

9.4.4.a Der Zoomfaktor

Bedingt durch die einfache mathematische Skalierbarkeit von Vektoren ist es

für Browser ein Leichtes alle darin vorkommenden Elemente proportional zu strecken beziehungsweise zu dehnen. Die gängige Technik ist bei den meistverwendeten Browsern wie *Internet Explorer* und *Chrome* jeweils das mittlere Mausrad.

Die folgende Abbildung 9.8 stellt einen stark vergrößerten Ausschnitt eines SVG Ausgabeartefaktes dar, um die Konzentration auf Detailebene zu veranschaulichen. Es können Objektverzerrungen im Mittel zwischen 500 und 25% realisiert werden was die meisten Benutzererwartungen erfüllen wird. Da diese Art der Skalierungsoperationen, im Sinne des *rendering*, auch einfach verarbeitbar sind, wird ein vergrößern in Echtzeit durchaus gewährleistet.

Den Navigationsmöglichkeiten innerhalb von SVG sind mit den Browsern noch lange keine Grenzen gesetzt. Es stehen diesbezüglich noch mächtigere Tools, wie zum Beispiel die frei verfügbare Software *Inkscape*[1], zur Verfügung die ein individuelleres *browsen* ermöglichen. Es steht, aufgrund der vielfältigen Werkzeuglandschaft sowie den flüssig verarbeitbaren Skalierungsoperationen, dem ergonomischen navigieren und somit dem gezielten Fokussieren auf Teilaspekten in großen Graphen nichts im Wege.

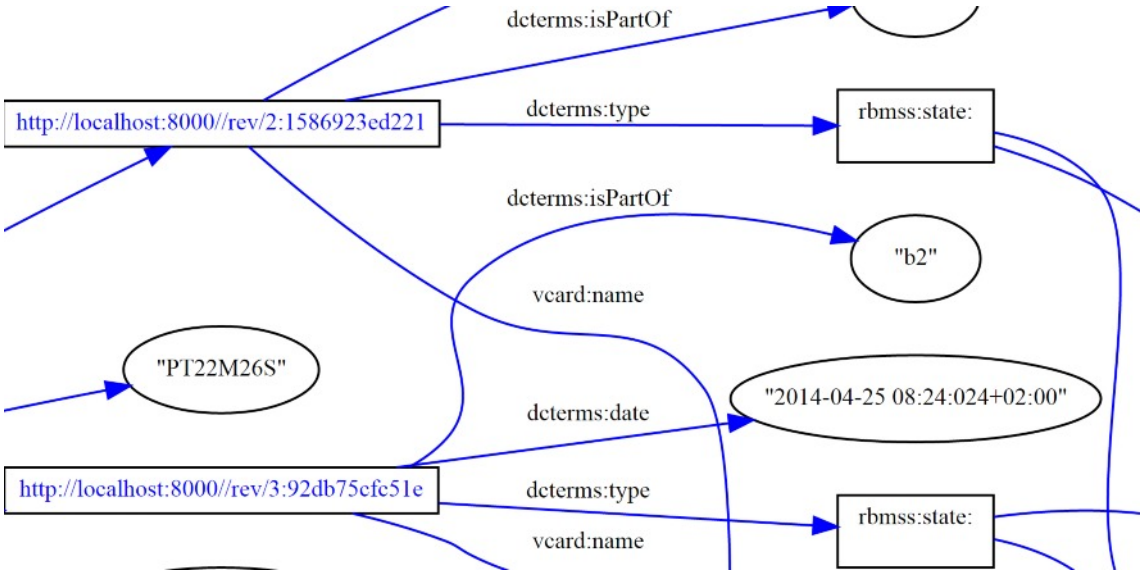


Abbildung 9.9 – ein stark vergrößerter Ausschnitt eines Projektsnapshots

9.4.4.b Die Tooltip Unterstützung

Um in einer Graphenstruktur logisch korrekt zu traversieren, spielt die Größe der Darstellung eine entscheidende Rolle. Die Praxis hat gezeigt, dass *Projektsnapshots* im Schnitt zirka 70 Knoten, die unterschiedlichster Informationsherkunft sind, besitzen. So gibt es ovale wie auch eckige Knotenformen, sogar welche, die durch die Ressourcenbezeichnung bedingt, in länglicher Form sind – alle drücken durch ihre Formgebung einen bestimmten Sachverhalt aus.

[1] http://www.inkscape.org/de/

So stellen ovale Knoten, die Blätter des Baumes dar und sind meist untypisierte Literale, sie entsprechen prinzipiell den Objekten im RDF-Graphen. Eckige Knoten hingegen sind primär typisierte Subjekte oft adressiert mittels einer URI-Referenz. Wie der stark vergrößerte Ausschnitt in obiger Abbildung 9.8 bereits andeutet, kann auch trotz der eindeutigen Graphennotation eine falsche Zuordnung irrtümlich interpretiert werden, da die Kantendichte ebenfalls ein Kriterium darstellt. Um auch hier nicht die Übersicht zu verlieren, bietet das SVG Format implizit eine sogenannte Tooltip Unterstützung an. Angemerkt sei hierbei, dass eine explizite Unterstützung des *tooltip* Attributs erst ab SVG Version 1.2 verfügbar ist, und diese Spezifikation befindet sich aber derzeit noch im *Draft-Zustand*[1]. Experimente haben gezeigt, dass zwar die meisten Browser es bereits unterstützen, jedoch unterschiedlich darstellen. Somit verfolgt RBMST den strikten Ansatz über den ratifizierten Standard SVG1.1, und bedient sich der impliziten Tooltip Hilfe. Genutzt kann sie werden indem der Mauszeiger eine Zeit lang über den Knoten verweilt, wo danach der Kanten-Knotenbezug explizit angeführt wird. Ein kleiner Auszug aus der SVG Spezifikation unterstreicht die weiteren Vorhaben des Kremiums:

„The SVG 1.1 language does not specify a method for the declarative display of tooltips on SVG content. While it suggests that the content of the title element could be displayed as a tooltip, it does not provide any control for the content developer. This meant that developers resorted to implementing their own tooltip functionality using scripting.

SVG 1.2 adds declarative tooltip support through the tooltip property. This allows for the display of textual tooltips to be declaratively controlled for each element in the SVG document. User agents should use the platform tooltip system if there is one available. On platforms without native support for tooltips, user agents can implement their own tooltip system. This means that the appearance, position and behavior of the tooltip is implementation or platform specific."[2]

Der Zoomfaktor sowie die Tooltip-Unterstützung die hier beispielhaft dargestellt wurden, stellen notwendige und unverzichtbare Techniken dar. Ohne sie würde, wie schon in Kapitel 1.5, den forschungsrelevanten Fragen, auch proklamiert, zwar die statische Informationsdarstellung weiterhin gewährleistet sein, jedoch nicht *effektiv,* da die notwendigen dynamischen Aspekte der Navigation im *Informationsraum des Graphen* fehlen würden.

[1] die Vorstufe einer verabschiedeten Standardversion (Specification)

[2] Zitat aus http://www.w3.org/TR/2004/WD-SVG12-20041027/applications.html

10 Kapitel

Die prototypische Forschungssoftware RBMST

"Das Ganze ist mehr als die Summe seiner Teile“
Aristoteles

Schwerpunkte in diesem Kapitel:

- Die Produktivversion RBMST
- Die Benutzerschnittstellen
- Die Artefaktausgabe

10.1 Die Produktivversion RBMST

In Kapitel 1.2 wurden eingangs Bestrebungen dargelegt die dieses Vorhaben im Grundsatz motivierten. Um nun die Gültigkeit dieser motivierten Aussagen auch erproben können bedingt es einem Werkzeug. Vor diesem Hintergrund entstand RBMST als eine mögliche Implementationsform basierenden auf der These von RBMSS.

10.1.1 Das standalone Executable

Die Vielzahl der zuvor diskutierten, kollaborativ entwickelten Anforderungen und auch technischen Bedingungen lies die notwendige Software-Architektur von Grund auf entstehen. Zur vollständigen Analyse des hier vorliegenden Software-Werkzeuges fehlt nun noch die abschließende Dokumentation des ausführbaren Programmes, samt den Benutzerschnittstellen und Artefakten aus der Sicht des potenziellen Benutzers.

10.1.2 Die Betriebssystemportabilität

Zuvor wurde schon die Portabilität als ein Hauptkriterium formuliert. Diese logische Anforderung ergibt sich aufgrund der vielfältigen Plattformabhängigkeiten der IT-Welt. Einmal ist Linux die Plattform der Wahl ein anderes Mal Windows oder MacOs. Aufgrund dieser Unvorhersagbarkeit, detektiert RBMST die Mercurial Applikation ohne Systempfadsuche und verwendet zur eigenen Konfiguration ausschließlich den XML-Konfigurationsdateimechanismus. Das Grafikframework, ScalaFX, adressiert ebenfalls diesen Aspekt, indem es die angebotenen Widgets[1] selbst emuliert und somit ein immer gleichbleibendes „Look & Feel“ ermöglicht

[1] http://de.wikipedia.org/wiki/Widget

[AG2009]. Das System wurde unter den Plattformen Windows XP, 7, 8 praxiserprobt bedarf jedoch sicherlich noch genauen Portabilitätstests für Linux & MacOS.

10.1.2.1 Synopsis

Das Programm RBMST in der hier dargelegten Version 1.0.0 wurde mit dem Entwicklungssystem Scala Version 2.11.2 entwickelt. Die darunterliegende Laufzeitumgebung ist JDK[1] in der Version 1.7.0_67, wobei die verwendeten Bibliotheken ausschließlich Komponenten der Java Runtime Engine sind, und somit ein dediziertes Development Kit nicht erforderlich machen. Als Werkzeug zum Bilden und Packen wurde die sehr innovative Build-Umgebung *sbt* Version 0.13.5 eingesetzt. Das RBMST-Packet kann mit dem sbt PlugIn *assembly* oder *pack*[2] erstellt werden, wobei letzteres aufgrund seiner Flexibilität vorgezogen wurde. Im Folgenden wird die zentrale Treiberdatei *build.sbt* vorgestellt, um die eigentlichen Bibliotheksabhängigkeiten und Compilereinstellungen darstellen zu können.

```
import AssemblyKeys._
assemblySettings
packAutoSettings
jarName in assembly := "RBMST.jar"
test in assembly := {}

name := "rbmst"
version := "1.0.0"
description := "Repository Based Management Support Tool"

licenses += "GPLv2" -> url("http://www.gnu.org/licenses/gpl-2.0.html")
organization := "vu.at.presulli"
organizationHomepage := Some(url("http://presulli.at.vu"))
mainClass in (Compile, packageBin) := Some("vu.at.presulli.thesis.rbmss.rbmst")

scalaVersion:="2.11.2"

scalacOptions ++= Seq(
"-unchecked",
"-deprecation",
"-feature" ,
"-encoding" ,
"iso8859-1")

resolvers ++= Seq(
"The Central Repository" at "http://search.maven.org/",
 "Scala-tools Maven2 Repository" at "http://scala-tools.org/repo-releases")
```

[1] Siehe dazu Java Development Kit: http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

[2] https://github.com/xerial/sbt-pack

```
libraryDependencies ++= Seq(
"org.scalafx" %% "scalafx" % "2.2.60-R9",
"org.scala-lang.modules" %% "scala-xml" % "1.0.2",
"org.scala-lang.modules" %% "scala-parser-combinators" % "1.0.2",
"org.scalatest" %% "scalatest" % "2.2.1",
"org.apache.jena" % "jena-arq" % "2.12.0",
"joda-time" % "joda-time" % "2.4",
"org.joda" % "joda-convert" % "1.6",
"org.jfxtras" % "jfxtras-labs" % "2.2-r5")

unmanagedJars in Compile +=
Attributed.blank(file(scala.util.Properties.javaHome) / "lib" / "jfxrt.jar")
exportJars := true //use jar file instead of classes directory
crossPaths := false    //remove the scala version in exported jar
fork in run := true    //run GUI in separate thread
testOptions in Test += Tests.Argument("-oD")    //show duration
```

Auflistung 10.1 – die zentrale Builddatei von sbt

Die wesentlichsten Punkte sind unter der Zuweisung scalacOptions sowie unter libraryDependencies zu finden, wobei andere Konstrukte wie zum Beispiel die Festlegung des Namensraumes und den Einsprungsspunkt der Applikation sicherlich auch von Interesse sind.

Zusammenfassend kann man festhalten, dass RBMST prinzipiell die scala eigenen Module scala-xml und scala-parser-combinators referenziert, wie auch joda-time für die Zeittransformationen und jfxtras für die Kalenderfunktionen der GUI. Das Apache jena Framework dient im weiteren Sinn der RDF-Graphen Erstellung, wobei die aus Sicht der Applikation genutzte eigens übersetzte Bibliothek *scardf* in einem separaten lib-Verzeichnis vorliegt. Die Vorteile von scalaFX wurden bereits diskutiert und werden deswegen anstelle von javaFX vorgezogen.

Anhand der Compilereinstellungen lässt sich erkennen, dass ein Hauptaugenmerk der Übersetzung, so wenige Spezialeinstellungen vorzunehmen wie nur möglich und sämtliche relevante Warnungen zu aktivieren, war. Es wurde auch explizit die Kodierungseinstellung iso8859-1 aktiviert um auch Umlaute der deutschen Sprache problemlos verarbeiten zu können [ME2006]. Es könnten andererseits unvorhersehbare Inkompatibilitäten in Interaktion mit dem Wirtsystem, dessen Kodierungsbasis falls nicht anders angegeben die native Einstellung des Betriebssystems ist, entstehen. Insgesamt ist die Applikation in Bezug auf 3rd Party Libraries und besondere Übersetzungsmodalitäten relativ anspruchslos. Besondere Systemanforderungen an das Grafiksystem oder an Speichersysteme sind ebenfalls nicht vorhanden.

10.2 Die Benutzerschnittstellen

Die Anforderungen an die Schnittstellen waren anfänglich als gering einzustufen. Im Zuge der kollaborativen Prozesse zur Konsensfindung stellten sich jedoch

die unterschiedlichsten Bedürfnisse zur Interaktion heraus. So war die anfängliche Kommandozeile bald an ihre Grenzen geraten und der Bedarf nach einer automatisierbaren Schnittstelle wurde geweckt.

Dem nicht genug, kam zuletzt noch der Wunsch nach mehr Ergonomie weniger der Optik wegen, sondern mehr im Sinne der vereinfachten menügeführten Bedienbarkeit. Ein graphisches Frontend, eine GUI war somit auch auf der Agenda. Insgesamt kann das System derzeit über drei separate Schnittstellen bedient werden, und mit der Systemsprache Englisch wurde auch der Fokus auf ein breites Anwendungspublikum gestellt.

10.2.1 Konsolenausgaben

Diese Form der visuellen Darstellung ist die Basalste. Sie stellt so gut wie keine Vorraussetzungen an das Betriebssystem und eignet sich daher auch sehr gut zur Fehlereingrenzung, sei es in Form von Entwicklungstests oder auch während der operativen Betriebsphase im Sinne von Systemwarnungen die eine nicht konforme Bedienung signalisieren.

10.2.1.a Warnungen

Da die Entwicklungstests, ansatzweise in Form von Test Driven Development [WD2008], die die *scalatest* Suite im Vorfeld verwendeten [OM2008] und jene die Standardausgabe an die Konsole leitet, wurde auch sie das Mittel der ersten Wahl.

Auflistung 10.2 zeigt beispielhaft ein paar Warnungen und Zustandsausgaben der Echtzeitausgabe:

```
Error in execution: Connection refused: connect
Notice: REVISION != ID in (12,14)
time delta(PT22M26S) + additional time(PT0S) = PT22M26S
Querying repository: localhost
"OutputPath" tag missing using default
 No relevant records found conforming to the querying parameter !
```

Auflistung 10.2 – beispielhafte Warnungs- und Fehlerausgaben

Die Ausgaben sind bewusst informeller Natur, sie geben keinen Fehlercode (*ErrorCode*) aus, der plattformspezifisch seitens des Benutzers decodiert werden müsste. Sie sollten dem normal informierten Benutzer zur Fehlereingrenzung genügen.

10.2.1.b Kurzhilfe

Die Konsole dient ebenfalls zur Ausgabe einer einfachen Anleitung. So wird bei Exekution des Programms *rbmst,* entweder ohne weiterer Parameter oder eines illegalen Kommandos, eine Beschreibung aller relevanten Kommandoargumente

über die Konsole in formatierter Weise ausgeben, wobei die dargestellten Optionen prinzipiell kommandoabhängig sind.

Werden bei den Kommandos *gui* und *bat* jeweils nur ein Argument benötigt, sind über den Modus *cmd* sämtliche Parameter anzuführen, auch jene wie *DotPath* und *OutputPath* welche in den anderen Modi obligatorisch in der XML-Konfigurationsdatei als Prolog vorhanden sind. Die nachfolgende Abbildung 10.1 listet die Kurzhilfe zu RBMST V1.0.0 auf.

```
rbmst {cmd|bat|gui} param1 ...
Synopsis:
This tool interrogates the destined repository with a given parameter set.
The set can be overgiven via cmd prompt, a xml configuration file or the gui interface.
The first argument is the command parameter, the second argument and above are dependent on the preceding command.
The following commands including arguments are supported:

cmd
      --DotPath->
      --OutputPath->
      --RepoPath->
      --Task->
      --Filter->
      --FilterAttribute->
      --ProjectAttribute->

bat <Path to Configuration File>

gui <Path to Configuration File>

Caveats:
If no ConfigurationFile is set, then the file "project-configuration.xml" in the working directory is assumed, cmd doesn't require it.
If no OutputPath is set, a directory named "output" will be automatically created in the working directory.
Appropriate permissions in the filesystem are required.
```

Abbildung 10.1 – Synopsis von RBMST auf Kommandozeilenebene

10.2.1.c Kommandozeile

Die Parametrierung über die Kommandozeile ist sicherlich die grundlegendste aller Interaktionsform, stellt aber zugleich auch die Schnittstelle mit den größtmöglichen Fehlerpotential dar, da die Anzahl der benötigten Parameter abhängig vom Bedarfsfall erheblich ist. Eine typische Argumentenparametrierung über die Kommandozeile soll hier in Auflistung 10.3 veranschaulicht werden:

--DotPath->C:\Progra~1\Graphviz
--OutputPath->ProjectArtefact
--RepoPath->C:\Projects\OCI
--Task->InterfaceDesign
--Filter->dependency=\w+
--FilterAttribute->useUTC : contact=author|person : completeInterval
--ProjectAttribute->SVGGraph:displayGraph:pack:encrypt=mypassword"

Auflistung 10.3 – die Parameter des Kommandozeilenaufrufes

Die Länge der Kommandozeile und Kontrollzeichen wie : und | stellen an den Benutzer eine anspruchsvolle Aufgabe. Hinzu kommt noch die richtige Parametrierung der gedachten Pfade, Filter und Attribute. Wie deutlich zu erkennen ist verwendet RBMST einen -> anstatt des klassischen = Zeichens für die Zuweisung um nicht mit den Angaben der Filterattribute zu kollidieren. Der Vollständigkeit halber werden nun in diesem Zusammenhang alle möglichen Parameterformen zwecks Dokumentation in tabellarischer Form zusammengefasst.

Die erste Hierarchie der Konfigurationsparameter ist in Tabelle 10.1 zusammengefasst, die weiteren Unterkategorien wie Filter, FilterAttribute und ProjectAttribute werden in den nachfolgenden Tabellen 10.2, 10.3 und 10.4 referenziert.

Parameter	*Typen*	*Instanzformen*	*Semantik*
DotPath	Pfad	C:\...	Pfad zum Graphviz Dot Werkzeug (dot.exe)
OutputPath	Pfad	C:\...	Pfad zur Artefaktausgabe dar und wird gegebenenfalls automatisch generiert
RepoPath	Pfad, URI	C:\... file:// http://	Pfad zum Repository
Author	String mit XML-Escapeing	John Doe	Name des Changset committers inklusive optionaler Kontaktadresse
Branch	WortString	Branch_1	Zweigname
Task	WortString	Task_1	Name des erdachten Aufgabenbereiches
DateFrom	DateString	2014-07-10 13:35:10	Intervallstart
DateTo	DateString	2014-08-12	Intervallende
Filter	Doppelpunkt separierte (:) Liste	siehe Tabelle 10.2	siehe Tabelle 10.2
FilterAttribute	Doppelpunkt separierte (:) Liste	siehe Tabelle 10.3	siehe Tabelle 10.3
Project Attribute	Doppelpunkt separierte (:) Liste	siehe Tabelle 10.4	siehe Tabelle 10.4

Tabelle 10.1 – Bedeutung der zentralen Parameter von RBMST

Parameter	*Typen*	*Instanzformen*	*Semantik*
Dependency	Java Pattern	Dependency=Task_1	Schlüsselpragmatik um eine Projektabhängigkeit zu injektieren*
State	Java Pattern	State=start|stop	Schlüsselpragmatik um einen Projektzustand zu injektieren
KnowledgeSource	Java Pattern	Knowledgesource=\w+	Schlüsselpragmatik um eine projektspezifische Wissensquelle zu injektieren
Enquiry	Java Pattern	Enquiry=\w+	Schlüsselpragmatik um eine Projektanfrage zu injektieren

Tabelle 10.2 – Bedeutung der Schlüsselpragmatikparameter von RBMST

* Falls kein Argument hinter dem = Zeichen angegeben ist, wird das werkseitige Java-Pattern *\w+* impliziert.

Die nachfolgenden tabellarisch angeführten Parameter dienen der Semantikeingrenzung über die Attributierung der zuvor dargestellten Filter, die die eigentlichen Schlüsselwortpragmatiken ergeben. Die Attribute werden automatisch im jeweiligen Kontext verwendet.

Die Projektattribute besitzen eine rein orthogonale Wirkungsweise. Sie sind von den Schlüsselwortpragmatiken unabhängig und wirken somit auf das angeführte Projekt gleichermaßen ein.

Parameter	*Typen*	*Instanzformen*	*Semantik*
useUTC	Wahrheitswert	useUTC	Falls gesetzt, Zeitanzeige in UTC
contact	String,Filter	contact=author contact=author|person contact=author|email	Abhängig vom Filter person, email werden dementsprechen die VCARD Elemente im Graphen dargestellt
completeInterval	Wahrheitswert	completeInterval	Falls gesetzt, wird bei der Schlüsselwortpragmatik state das volle Interval miteinbezogen
useCostMetric	Double	useCostMetric=10	Der Kostenmetrikfaktor

Tabelle 10.3 – Bedeutung der schlüsselwortspezifischen Parameter

Parameter	*Typen*	*Instanzformen*	*Semantik*
SVGGraph	Wahrheitswert	SVGGraph	Falls gesetzt, Generierung von SVG andernfalls RDF
displayGraph	Wahrheitswert	displayGraph	Falls gesetzt, automatisches öffnen der Datei mit Standartapplikation
pack	Wahrheitswert	pack	Falls gesetzt, komprimieren mittels GZip-Kompressor
encrypt	String	encrypt=password	AES 128Bit Verschlüssselung des Inhalts mit 16 stelligen Passwort*

Tabelle 10.4 – Bedeutung der projektspezifischen Parameter

* Das Passwort wird automatisch auf 16 Stellen mit Leerzeichen ausgerichtet

10.2.2 Die Stapelverarbeitung

Diese Schnittstellenart ist sicherlich die gebräulichste, da sie einfach zu bedienen ist und etablierte Anfragen an das Repository sich im Grunde nur sehr selten ändern. Ein Verwalten mehrerer Stapeldateien ist ebenfalls unproblematisch. Falls eine Stapeldatei zur Ausführung kommt, werden alle darin aufgelisteten Projekte seriell abgearbeitet. Wichtig hierbei ist, dass bei mehreren Projektangaben der jeweilige Projektname stets variiert, da ansonsten Namenskonflikte in der Benennung der Ausgabedateien entstehen. Anderenfalls würde letztes gleichnamige Projekt das vorhergehende Ausgabeartefakt überschreiben.

Dies Form der Stapelverarbeitung ist sehr beliebt, da sie in einfacher Weise automatisiert eingebunden, und zur weiteren Instanzierung als Schablone verwendet werden kann. Ein einfaches ent- beziehungsweise auskommentieren von Filter- und Projektkriterien ist problemlos möglich und erlaubt somit eine schnelle Adaption der gewünschten Parametersätze. Untenstehende Abbildung 10.2 führt zwei Projekte mit unterschiedlichen Adressierungen zur sequentiellen Abarbeitung an.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<Configuration>
<DotPath>C:\Program Files\Graphviz</DotPath>
<OutputPath>C:\Users\Beata\Deskto~1\output</OutputPath>
<!--Project Configurations-->
      <!--<Project RepoPath="http://localhost:8000/Driver">-->
      <Project RepoPath="C:\Users\Beata\Deskto~1\Project1">
      <ProjectAttribute>SVGGraph:displayGraph:pack:encrypt=mypassword</ProjectAttribute>
            <Author>Beata Scherbantin &lt;b.scherbantin@xxx.com&gt;</Author>
            <Task>PanelSteering</Task>
            <Branch>GUIBranch</Branch>-->
            <DateFrom>2014-04-12 00:00:00</DateFrom>
            <DateTo>2014-08-12 10:18:29</DateTo>
            <Filter>state=\w+ : enquiry=\w+ : knowledgeSource=\w+: xxx=\w+ :dependency=\w+ :demo=\w+</Filter>
            <FilterAttribute>useUTC : contact=author|person : completeInterval : useCostMetric=10 </FilterAttribute>
  </Project>
  <Project RepoPath="http://localhost:8000">
            <!--<Task>Me1600</Task>-->
            <!--<Author>rbe</Author>-->
            <DateFrom>2014-02-25 13:18:29</DateFrom>
            <DateTo>2014-12-06 13:18:29</DateTo>
            <ProjectAttribute>native : pack</ProjectAttribute>
  </Project>
</Configuration>
```

Abbildung 10.2 – selektives Aus- und Entkommentieren von Projektparametern

10.2.2.a Rekursive Exploration

Ein weiterer Effekt der positiv genutzt werden kann ist die rekursive Abarbeitung verschachtelt verwalteter Projekte. Es gibt, abhängig von der Projektkomplexität und auch den entwicklungsstrukturellen Vorgaben, manche SubProjekte die innerhalb eines verwalteten Projektbaumes gesetzt werden. Mercurial bietet dazu einen eigenen Mechanismus über das 3rd Party Plugin *Guestrepo*[1] an. Ein Beispiel

[1] https://bitbucket.org/selinc/guestrepo

dazu wäre: Ein Hauptprojekt benutzt weitere Unterprojekte die aber von externen Entwicklungen gewartet, und somit nicht direkt in das Projekt kopiert werden, da die externe Codebasis noch nicht stabil genug ist.

In diesem Falle greift bewusst der rekursive Prozess der Abarbeitung ein. Die baumartige Pfadwurzel wird rekursiv bis zum letzten Blatt traversiert, und bei Vorhandensein eines gültigen Repositories, mit den gleichen Filtereinstellungen wie das Masterprojekt selbst exploriert. Damit ergibt sich eine implizite Darstellung des Unterprojektes, sofern Schlüsselwortpragmatiken verwendet wurden, die jedoch zuvor auf konventionelle Weise zwischen den Beteiligten abgeklärt werden sollten.

10.2.3 Grafische Benutzerschnittstelle

Diese Art der Interaktion stellt sicherlich eine der komfortabelsten dar. Sie ist ein 1:1 Abbild der Kommandozeilenparametrierung jedoch entworfen mit Grafikelementen vorwiegend bestehend aus Checkboxen und Radiobuttons die so gut wie keine manuelle Texteingabe mehr erfordern. Sie stellt bezüglich Software-Ergonomie zugleich die sicherste Eingabeform dar.

Gleichzeitig wird durch graphische Abbildung der derzeit implementierten Schlüsselpragmatiken auch eine gewisse Limitierung gesetzt. Falls die Menge der Schlüsselwortpragmatiken erweitert werden sollte, muss auch hier entwicklungstechnisch eingegriffen werden und die Schnittstelle dementsprechend harmonisieren, was im Beispiel der Stapelverarbeitung keine Notwendigkeit darstellt.

Die werksseitigen Voreinstellungen entsprechen prinzipiell auch jenen der anderen Schnittstellen, bis auf die Ausnahme, dass alle Schlüsselwortpragmatiken per default aktiviert sind. Logikverschränkungen verhindern, sofern möglich, durch gezieltes Verriegeln der Optionen eine unbeabsichtigte Fehleingabe.

Ebenfalls bemerkenswert ist das plattformunabhängige „Look & Feel“ der Grafikelemente. Es ist für den gewöhnlichen Benutzer von Vorteil wenn er in seiner gewohnten graphischen Umgebung so wenige Veränderungen wie möglich vorfindet. Da die Auswahl des Entwicklungssystems stets den Projektanforderungen folgt, ist es somit sinnvoll ein Widget-Toolkit [SR2014] zu verwenden, das die Grafikelemente vorwiegend selbst emuliert um auch immer den gleichen graphischen Effekt zu erzeugen.

Nachteilig zu erwähnen wäre der Aspekt der Performance, der sich aber nur in Extremfällen wie bei rasanten Spielen bemerkbar machen würde, da dort die native Grafikunterstützung ausbliebe, wobei auch das wiederum nur bedingt richtig ist. Neuerdings verwendet javaFX ab der Version 2.0 und somit auch das Framework scalaFX bereits teilweise die Hardware Beschleunigung ausgewählter Grafikkartenchipsets. Wie auch immer, diese Anforderung an eine erhöhte Grafikleistung ist im Fall von RBMST, bedingt durch seine rein statische Eingabemaske, definitiv nicht gegeben. Die folgende Abbildung 10.3 verdeutlicht die initiale Standartoberfläche.

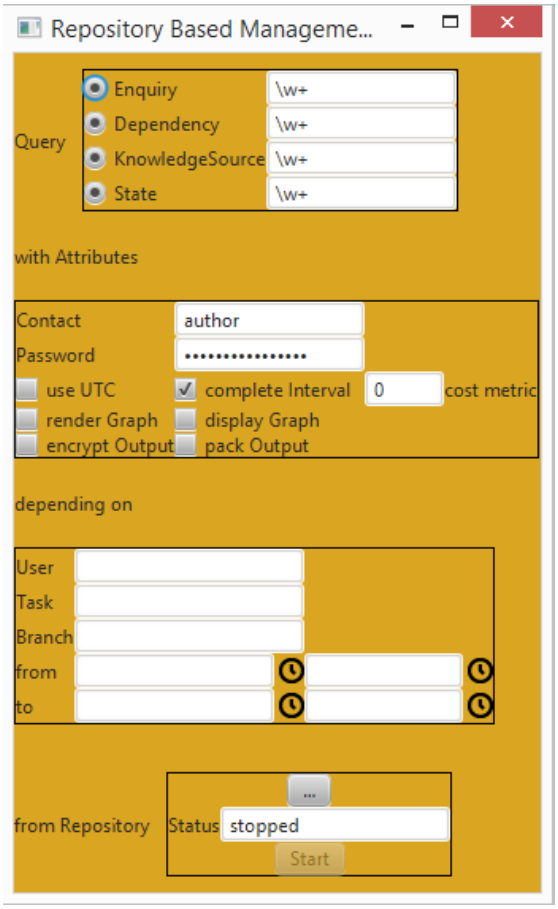


Abbildung 10.3 – die graphische Bedienoberfläche von RBMST

10.3 Die Artefaktausgabe

Das Resultat einer jeden Informationsextraktion wird in Form von Artefakten ausgegeben. Sie entsprechen der Serialisation eines Graphen sei es in Form von RDF/XML, oder auch als graphische Repräsentation mittels des skalierbaren Vektorgrafikformates SVG. In beiden Ausgabeartefakten ist der semantische Inhalt stets derselbe. Die Darstellung in SVG ist eher für die menschliche Interpretation geeignet, während das andere Format einer maschinellen Bearbeitung leichter zugeführt werden kann. Beide Artefakte behalten so ihren Anspruch auf eigenständige Interpretation, und können auch unabhängig voneinander erzeugt werden, wobei die SVG Generierung stets eine RDF/XML Serialisation als Transformationsbasis benötigt.

Es wird im nächsten Abschnitt eine minimale Informationsextraktion beispielhaft wiedergegeben, da ansonsten ein zu großer Datenumfang die Übersichtlichkeit ungünstig beeinflussen würde. Die folgende RDF/XML Auflistung unter 10.3.1 sowie Abbildung 10.4 stellen jeweils idente Informationsrepräsentationen desselben Metadatenbestandes dar. Die Unterschiede in Form der Interpretation werden sofort offensichtlich, dennoch werden die resultierenden Graphen und ihre semantischen Zusammenhänge zwecks Vergleichsmöglichkeiten ausführlich dargestellt. Tabelle 6.1 stellt mit ihren Typ-Deklarationen eine hilfreiche Referenz dar.

10.3.1 RDF-Datei

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/#"
  xmlns:rbmss="http://purl.org/rbmss/terms/1.0/#"
  xmlns:vcard="http://purl.org/vcard/elements/1.1/#" >
 <rdf:Description rdf:about="http://localhost:8000//rev/1:4f3322c412a0">
  <dcterms:date>2014-09-23 13:19:051+02:00</dcterms:date>
  <dcterms:type rdf:nodeID="A0"/>
  <dcterms:isPartOf>default</dcterms:isPartOf>
  <vcard:name>mpr</vcard:name>
 </rdf:Description>
 <rdf:Description rdf:nodeID="A1">
  <rdf:type rdf:resource="http://purl.org/rbmss/terms/1.0/#state"/>
  <dcterms:text>start</dcterms:text>
  <dcterms:isPartOf>Task1</dcterms:isPartOf>
 </rdf:Description>
 <rdf:Description rdf:about="http://purl.org/rbmss/terms/1.0/#duration">
  <dcterms:SizeOrDuration>PT3M35S</dcterms:SizeOrDuration>
 <dcterms:references rdf:resource="http://localhost:8000//rev/2:7d63f4cf6141"/>
 <dcterms:references rdf:resource="http://localhost:8000//rev/0:c09fe93e7d91"/>
  <dcterms:extent>0.5972222222222222</dcterms:extent>
 </rdf:Description>
 <rdf:Description rdf:nodeID="A0">
  <rdf:type rdf:resource="http://purl.org/rbmss/terms/1.0/#knowledgeSource"/>
  <dcterms:text>Reference_1</dcterms:text>
  <dcterms:isPartOf>Task1</dcterms:isPartOf>
 </rdf:Description>
 <rdf:Description rdf:about="http://localhost:8000//rev/2:7d63f4cf6141">
  <dcterms:date>2014-09-23 13:21:031+02:00</dcterms:date>
  <dcterms:isReferencedBy>tip</dcterms:isReferencedBy>
  <dcterms:type rdf:nodeID="A2"/>
  <vcard:name>mpr</vcard:name>
  <dcterms:isPartOf>default</dcterms:isPartOf>
 </rdf:Description>
 <rdf:Description rdf:about="http://localhost:8000//rev/0:c09fe93e7d91">
  <dcterms:isPartOf>default</dcterms:isPartOf>
  <dcterms:date>2014-09-23 13:17:056+02:00</dcterms:date>
  <vcard:name>mpr</vcard:name>
  <dcterms:type rdf:nodeID="A1"/>
 </rdf:Description>
 <rdf:Description rdf:nodeID="A2">
  <dcterms:text>stop</dcterms:text>
  <dcterms:isPartOf>Task1</dcterms:isPartOf>
  <rdf:type rdf:resource="http://purl.org/rbmss/terms/1.0/#state"/>
 </rdf:Description>
</rdf:RDF>
```

10.3.2 SVG-Datei

Wie man bereits in der gedrehten Abbildung 10.4 erkennen kann, ist die graphische Darstellung im Vergleich zur vorherigen RDF/XML Serialisation, in Bezug auf die Übersichtlichkeit, deutlich überlegen und stellt auch deswegen eine klare Präferenz dar.

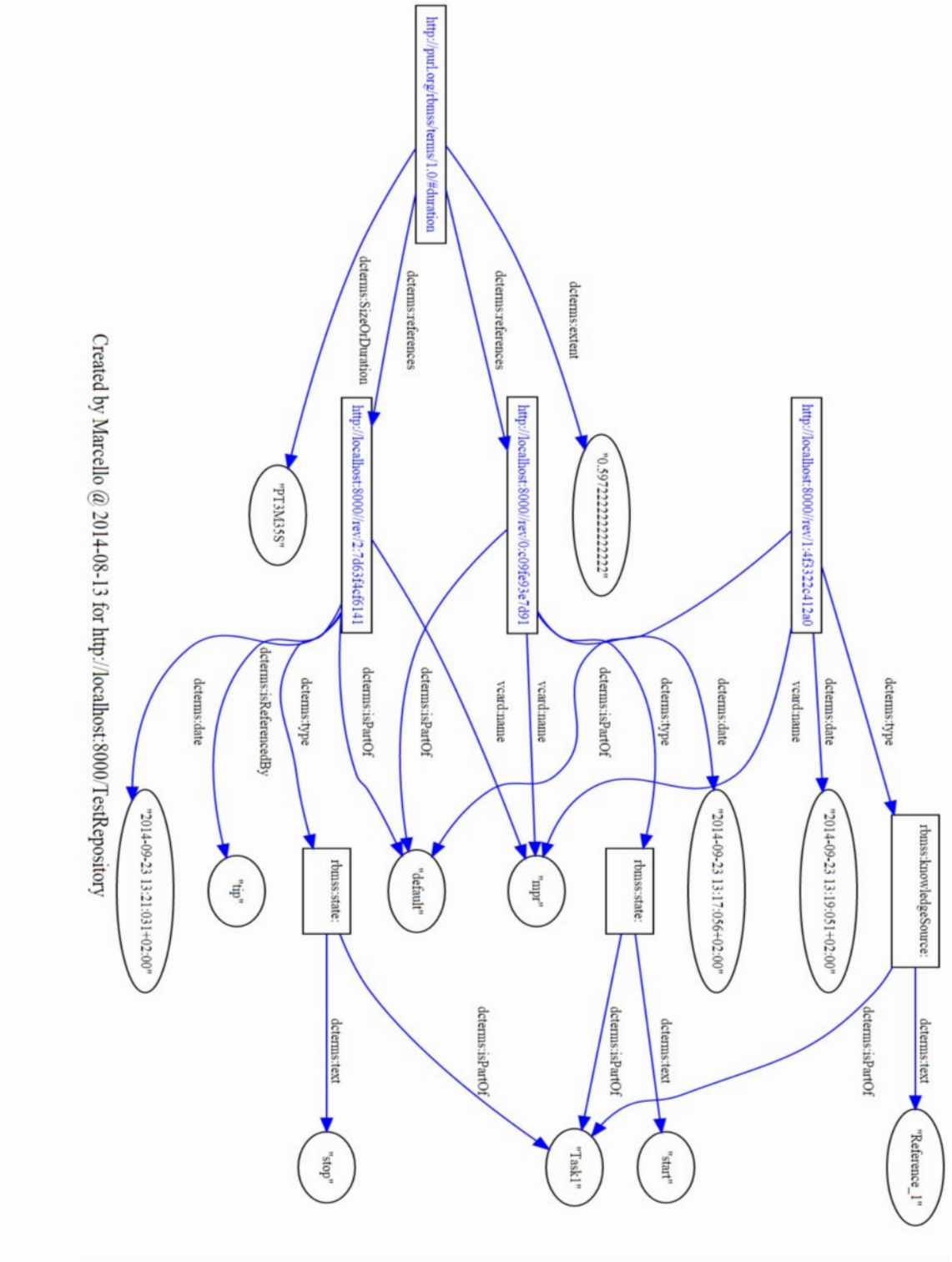


Abbildung 10.4 – ein vollständiger Projektgraph in SVG Serialisation

Teil IV
Die Erprobungsphase und deren Auswertungen

11 Kapitel

Ermittlungsmethoden und die Wahl der Industriepartner

"If you can not measure it, you can not improve it."
Lord Kelvin

Schwerpunkte in diesem Kapitel:

- Die Wahl der empirischen Ermittlungsmethode
- Der Feststellung des Unternehmensnutzen
- Die Industriepartner

11.1 Die Wahl der empirischen Ermittlungsmethode

Bezogen auf die Aussagen der These steht eine adäquate Erprobungsphase gegenüber. Sie dient der prinzipiellen Prüfung der Annahmen die eingangs in informeller Weise dargestellt wurden, jedoch bereits im Zuge der Motivationsdarstellung in Kapitel 1.3 als Hypothesen reformuliert.

Um nun die interessanten Aspekte in der Praxiswelt validieren zu können gibt es mehrere Möglichkeiten. Es könnte zum Beispiel eine Langzeitstudie vereinbart werden, wo dann Unternehmen sich dementsprechend dazu vertraglich verpflichten würden, oder eine kürzere aber intensivere Studie die ausschließlich auf heterogene IT-Projekte und taktische Phasen einer ausgewählten Anzahl an Partnerunternehmen ausgerichtet wäre.

Der Weg der für diese Erprobungsphase gewählt wurde war der letztere. denn es erschien sinnvoller eine Forschungssoftware nicht einer Langzeiteinsetzbarkeit zu unterwerfen, sondern vielmehr an den Anforderungen unterschiedlichster Unternehmens- und Projektstrukturen zu messen.

Eine Langzeitstudie würde auch einen vorvertraglichen und kommerziellen Aspekt mit sich führen, der auch sicherlich den Anspruch auf eine Software mit dediziertem Support hätte, der so nicht haltbar wäre. Vorteilhaft wäre jedoch sicherlich eine ausgereiftere Austestung des Werkzeuges mit seinen kollaborativ entwickelten Schlüsselwortpragmatiken. Nachteilhaft hingegen wäre das lange Warten auf jene Phasen wo Projektstatus- und Projektabgleichsmeetings kumulieren, um die erwarteten Effekte studieren zu können.

Eine Kurzzeitstudie hat vor allem den Vorteil der schnellen Reaktion auf technische Änderungswünsche und auf eine Fokussierung eines relevanten Phasenabschnittes, in agiler Methodenformulierung eines sogenannten *Sprints*. In dieser kurzen Zeit verdichten sich die Informationsbedürfnisse und der eigentliche Nutzen tritt sofort zu Tage.

Ein weiterer Pluspunkt, der sich durch die vertragslos vereinbarte Kurzzeitstudie einstellt, ist die Unverbindlichkeit. Es ist davon auszugehen, dass wesentlich mehr Unternehmen zumindest indifferent oder gar geneigt sind zu kooperieren als im anderen vorvertraglichen Fall. Dies impliziert wieder einen gewissen Freiheitsgrad in der Auswahl der Unternehmen samt der Projekte im Hinblick auf den Skalierungsgrad, der eine weit wesentlichere Rolle spielt als oft angenommen. Man kann in diesem Zusammenhang auch von einer Querschnittsstudie sprechen.

Sind es doch die heterogenen Projektlandschaften und die hohe Dispersion der Unternehmensstrukturen die in der Lage sind, die wahre Untermauerung einer These tatsächlich absichern zu können. Anders formuliert, bezogen auf das zuvor formulierte Modell, besitzt eine breitbandige Kurzzeitstudie mehr an Effektivität als eine schmalbändige Langzeitstudie, wobei die Stabilität der Erkenntnis, sowie das gesicherte Wissen, kann mit einer Langzeitstudie natürlich nicht konkurrieren. Nachfolgende Tabelle stellt markante Unterschiede der jeweiligen Studienform dar.

Eigenschaft	*Kurzzeitstudie*	*Langzeitstudie*
Laufzeit	Gering	Hoch
Persönlicher Zeitaufwand	Gering	Hoch
Materieller Ressourcenbedarf	Gering	Hoch
Stabilitätsanforderung des Testumfeldes	Gering	Hoch
Vertragliche Kontrahierungsaspekte	Gering	Hoch
Aufwand der Datensicherung	Gering	Hoch
Statistische Relevanz	Gering	Hoch
Erkenntnisgewinn durch Feldforschung	Gering	Hoch
Beobachtungen von Langzeiteffekten	Gering	Hoch
Varianz der Resultate	Hoch	Gering
Fokus auf Querschnittsaspekte	Hoch	Gering
Modellanpassung durch schnellere Iteration	Hoch	Gering

Tabelle 11.1 – Vergleich zwischen Kurz- & Langzeitstudie

Prinzipiell sind beide Studienformen anwendbar und auch sinnvoll, den möglichen Einsatz entscheiden rein die realen Gegebenheiten. Die hauptsächlichen Beweggründe für die Wahl einer Kurzzeitstudie mit Querschnittsaspektierung waren, der hohe persönliche Zeitaufwand, die Restriktion allfälliger Ressourcen, vorvertragliche Kontrahierungsaspekte sowie die Möglichkeit der schnelleren Erkenntnisableitung.

11.2 Die Feststellung des Unternehmensnutzen

Es wird der Versuch unternommen einen so weit wie möglich objektiven Nutzen durch zwei getrennte Erfassungsmethoden, in quantitativer und qualitativer Hinsicht, festzustellen. Zu diesem Zweck werden für die quantitative Ermittlung zwei verschiedene Fragebögen erstellt. Der beiden Bögen, der eine technisch orientiert, der andere auf Management zentriert, verfolgen dasselbe Ziel – mittels der gewonnen Stichprobe Rückschlüsse auf die Grundgesamtheit zu ziehen. In beiden Fällen wird durch eine neutrale Fragestellung eine Querschnittsbetrachtung aller relevanten Aspekte unternommen, wobei die jeweiligen Fragen gemischt positiv wie auch negativ formuliert werden um die Aufmerksamkeit der Befragten ungeteilt zu erhalten. Die maximale Anzahl der Fragen sollte laut [GL1999] nicht über 15 bis 25 Fragen liegen beziehungsweise maximal 10 Minuten dauern, da ansonsten durch die zeitliche Überbeanspruchung der Wahrheitsgehalt der Antworten rapide abnimmt. Mittels dieser evaluativen Teilstudie wird das Unternehmen in jenem Maße eingebunden, dass der reguläre Betriebsablauf nicht *intrusiv* gestört wird, was durch ein zeitliches Überbeanspruchen sicherlich der Fall wäre. Durch die unternehmensweite Querschnittsbetrachtung eines konkreten Projektablaufes können sich aus Unternehmersicht auch Verbesserungspotentiale abzeichnen. Sofern, im Sinne einer Performancesteigerung [BJ2008] erfolgreich umgesetzt, können sich dadurch Wettbewerbsvorteile einstellen.

11.2.1 Quantitative-Messung

Die in Abbildung 15.1 & 15.2 verfassten Fragebögen dienen prinzipiell zur elektronischen Erfassung quantitativer Phänomene. Jene Fragen besitzen teilweise rein numerische Felder, andere wiederum, die Neigungen und allgemeine Trends erörtern sollen, werden mittels einer Ja/Nein Entscheidungsfrage oder einer 4 Noten Bewertungsskala nach *Likert* zur detaillierteren Auswertung erfasst [MH2003]. Der Formulierung *sehr - nicht* steht eine gerade Notenanzahl von 4 gegenüber, damit keine indifferente Einstellung ausgedrückt werden kann. Der Fokus ist auf eine rein positive, beziehungsweise negative Aspektion gerichtet, um im weiteren Verlauf Tendenzen hinsichtlich einer möglichen Teilanpassung von RBMST besser ableiten zu können.

11.2.2 Qualitative-Messung

Die qualitative Erfassung, in Abbildung 15.3 & 15.4 in Form von Interviewfragen, beschäftigt sich hauptsächlich mit subjektiven Eindrücken der beteiligten Individuen. Es werden hierbei verbale Antworten erfasst, Beobachtungen in der Feldforschung sowie in der Gruppendiskussion angestellt. Diese Technik ist informeller Natur, zeigt jedoch interessante Intentionen und Motivationen auf, die in der quantitativen Bewertung nur sehr schwer bis gar nicht erfasst werden können. Besonders gut eignet sich diese Analyse um subtile subjektive Handlungszusammenhänge wie auch jene in den Gruppenprozessen besser verstehen und bewerten zu können.

11.2.3 Das Erhebungsverfahren

Die Beantwortungen der Fragebögen werden völlig elektronisch anonym erfasst und anschließend mittels statistischer Lagemaße ausgewertet. Es handelt sich um den Modus der moderierten Bildschirmauswertung, der mittels der universitären Fragebogensoftware *GRAFSTAT4*[1] Version 4.298 professionell generiert und auch durchgeführt werden kann.

Die Rohform der Fragebögen sowie die Struktur der immer gleich gearteten Interviewfragen stehen in Kapitel 12.1.2 zur weiteren Verfügung. Nach Abschluss der Befragung wird von den jeweiligen Unternehmen ein Manifest Vorort erstellt, das den Einsatzrahmen der exerzierten Erprobungsphase sowie die abgehaltenen Interviews dokumentiert und somit deren Ursprung und Geltungsbereich nachvollziehbar macht.

Die Gruppen werden stets getrennt nach ihrer funktionalen Verwendung befragt um ungewollte Überschneidungen zu vermeiden. Nach der Fragebogenbeantwortung werden selektive Interviews geführt, wobei die Auswahl der konkret Befragten nicht vorher bekanntgemacht wird [PA2002]. Diese Informationen werden mit den gewonnen Erfahrungen aus der Feldforschung verknüpft umso ein einheitliches Gesamtbild entwerfen zu können. Nachstehende Abbildung 11.1 zeigt die vielfältigen Einsatzmöglichkeiten der Fragebogensoftware auf.

Abbildung 11.1 – der Startbildschirm von Grafstat

[1] http://www.grafstat.de/bezugsquellen.htm

11.3 Die Industriepartner

Im Folgendem werden die vier beteiligten Unternehmen kurz vorgestellt und gemäß ihrer jeweiligen historischen Entstehungsgeschichte und organisatorischen Struktur charakterisiert um den theoretischen Skalenfaktor der Testbandbreite erfassen zu können. Nach jeder Unternehmensbeschreibung folgt eine statische Projektbeschreibung die die wesentlichsten Faktoren wie Projektname, Art und Umfang wie auch die eingesetzten IT-Supportsysteme anführt.

Dieser schließt sich der dynamische Aspekt des Projektverlaufes im Sinne der gelebten Ablaufstruktur und eine explizite Darstellung der manch implizit vorherrschenden Vorgehensmodelle innerhalb der konkreten Softwareentwicklung an, wobei sich die Darstellung nur auf relevante Projektaspekte konzentriert. In Kapitel 13.1.3 wird nochmals auf jeweilige Unternehmenseigenheiten eingegangen, jedoch verstärkt in Bezug auf etwaige strukturelle Änderungen während der Projektdurchführungsphase sowie interessante Querschnittsbeobachtungen und jene die innerhalb der Gruppendiskussionen angestellt werden konnten.

Die folgenden Unternehmen werden taxativ gemäß ihrer Unternehmensgröße aufsteigend vorgestellt, wobei auf einheitlichen Wunsch beteiligte Personen bis auf die Geschäftsführung beziehungsweise Führungskräfte größerer Organisationseinheiten wie zum Beispiel involvierte Abteilungen *anonymisiert* behandelt bleiben.

11.3.1 Firma LEGIOKILL

Das Unternehmen[1] wurde 2009 in Form einer Kapitalgesellschaft mit beschränkter Haftung (GmbH) mit Unternehmenssitz in Wien gegründet. Als Unternehmensgegenstand befasst sich die Unternehmung vorwiegend mit der Entwicklung intelligent prophylaktisch agierender Systeme zur Bekämpfung der Legionellenbakterien[2], die die Ursache der lebensbedrohlichen Legionärskrankheit darstellen. Ein weiteres Geschäftsfeld welches 2011 eröffnet wurde, widmet sich generell der Wasserhygiene und der Wassereinsparung.

11.3.1.a Die Unternehmensstruktur

Die Unternehmung umfasst insgesamt weitere 8 Mitarbeiter. 2 Personen sind ausschließlich für den Vertrieb und die Marktanalysen tätig, 2 weitere Mitarbeiter sind für die Administration und Verwaltung zuständig. Des Weiteren gibt es auch einen Produktmanager und zwei Techniker die die Entwicklung besetzen. Für die Montage und Wartung ist noch eine weitere Person beschäftigt.

11.3.1.b Die Projektcharakteristik

Das gegenständliche Projekt, betitelt mit den Namen *Legio-Manager,* ist eine Software für den Endbenutzer des Systems LegioKill®. Sein Haupteinsatzzweck dient der Konfiguration, Darstellung und Erfassung von Parametern und Telemet-

[1] http://www.legiokill.at/

[2] http://de.wikipedia.org/wiki/Legionellen

riedaten die im System LegioKill persistent gespeichert sind. Die Datensätze sind beispielsweise Temperaturwerte, Spülintervalle oder auch Systemalarme. Die gesamten Elemente können vom Endbenutzer zentral vom System aus mit der Legio-Manager Software gemanagt werden. Die Software ist im Kern eine MS-Windows basierte *rich Clientapplikation* die über herkömmliche Browser wie den Internet Explorer bedient werden kann.

Das zentrale FrontEnd bildet ein WebServer welcher über einen proprietären Kommandoprozessor, im Sinne des Model-View-Control Modells, Konfigurations- und Visualisierungssequenzen darstellt und bedienbar macht. Eine im BackEnd befindliche Datenbank verwaltet die Einstellungsparameter pro gemanagten System. Die untere Treiberschicht ist in C kodiert, alle anderen Schichten sind aufgrund der holistischen Entwicklungsumgebung in Erlang implementiert.

11.3.1.b.1 Die projektspezifische IT-Landschaft

In diesem Projekt kommt als VCS, Mercurial in der Version 2.9 zum Einsatz. Als Entwicklungsplattform wird ausschließlich Windows 7 eingesetzt. Bugzilla wird zur Fehlerverfolgung jedoch rein intern verwendet. Entwicklungsinternes Wissen wird als einfacher Dateikatalog verwaltet. Die projektspezifische Kommunikation läuft strikt über Email.

11.3.1.c Die Projektdurchführung

Das Projekt Legio-Manager läuft bereits seit Februar 2013 und befindet sich derzeit in der Version 2.9.99. Es wird von den 2 Entwicklern ständig und dem einen Produktmanager in Teilzeit bearbeitet, sie stellen das eigentliche *Projektteam* dar. Der Produktmanager erfasst die Kundenanforderungen indirekt über die Vertriebsschiene. Die gewünschten Leistungsmerkmale werden intern mit dem Geschäftsführer diskutiert und abgewogen. Die tatsächlich erwählten Neuanforderungen werden dann vom Projektmanager mit den Entwicklern abgeschätzt und ein ungefährer Fertigstellungszeitpunkt dem Vertrieb bekanntgegeben.

11.3.2 Firma METAMAGIX

Das Unternehmen[1] wurde 2001 in Form einer Kapitalgesellschaft mit beschränkter Haftung (GmbH) mit Unternehmenssitz in Wien gegründet. Als Unternehmensgegenstand befasst sich die Unternehmung vorwiegend mit der Entwicklung Java-basierter Desktopsysteme die hauptsächlich eine Ausrichtung auf Business Applikationen mit Webtechnologien und auf das Management verschiedenster Domänen wie Portfoliomanagement oder Workflowverwaltung besitzen. Ein weiteres Geschäftsfeld widmet sich generell dem Consulting (Beraten) hinsichtlich Einsetzbarkeit diverser Systeme, von Datawarehousing bis hin zu CRM[2].

[1] http://www.metamagix.at/

[2] Customer Relationship Management

11.3.2.a Die Unternehmensstruktur

Die Unternehmung umfasst insgesamt weitere 10 Mitarbeiter. 2 Personen sind ausschließlich für den Vertrieb und die Marktanalysen tätig, 2 weitere Mitarbeiter sind für die Administration und Verwaltung zuständig, die restlichen 6 Personen stellen das komplette Entwicklungsteam.

11.3.2.b Die Projektcharakteristik

Das gegenständliche Projekt, betitelt mit den Namen *NQZ,* ein Akronym welches für *Nationales Qualitätszertifikat für Alten & Pflegeheime* steht, ist eine Software für den Endbenutzer eines Workflowsystems. Sein Haupteinsatzzweck dient der persistenten Verwaltung, Abbildung, und intelligenten Workflowsteuerung hinsichtlich einer Qualitätszertifizierung gemäß §20a Bundes-Seniorengesetz unter besonderer Berücksichtigung spezieller Anforderungen von Alten & Pflegeheimen.

Die Software ist im Kern eine plattformunabhängige Anwendung basierend auf dem offenen Document Management Systems *Alfresco*. Die Entscheidung fiel auf das Java-basierte Framework, da es mächtig, gut erweiterbar und anpassbar ist, auch die Zugriffssteuerung des DMS[1] funktioniert ähnlich wie bei gewöhnlichen Laufwerken.

Das zentrale FrontEnd bildet eine Weboberfläche mittels derer über eine intelligente Workflowlogik begleitende Arbeitsabläufe bezüglich einer anstehenden Zertifizierung abgebildet werden. Eine im BackEnd befindliche Datenbank persistiert die Prozesse pro Zertifizierung. Die Workflowlösung bedient sich ebenfalls größtenteils dem *Alfresco* BPM-Workflowkonzept. Das Nationale Qualitätszertifikat besteht aus sechs Phasen, wobei jede Phase beinhaltet separate Zugriffsrechte, Dokumentenerstellungsbereiche und Optionen der Benachrichtigung.

11.3.2.b.1 Die projektspezifische IT-Landschaft

In diesem Projekt wird als VCS Mercurial, in der Version 3.0.2, verwendet. Als Entwicklungsplattform kommen Windows 7 und auch Ubuntu-Linux in einem Mischbetrieb zum Einsatz. Die RedMine Plattform wird hier verstärkt eingesetzt. Sie wird zur Fehlerverfolgung und Trouble-Ticketing herangezogen und bietet auch eine externe Schnittstelle seinen Kunden an. Entwicklungsinternes Wissen wird über das integrierte Wiki der Plattform persistiert. Die projektspezifische Kommunikation läuft klassisch über Email und auch über Ticketing ab.

11.3.2.c Die Projektdurchführung

Das Projekt NQZ läuft bereits seit Juni 2013 und befindet sich derzeit in der Produktivversion 2.0.0. Es wird von 2,5 Entwicklern betreut wobei die halbe Kraft sich mehrheitlich auf die Projektkoordination konzentriert, sie stellen somit das eigentliche *Projektteam* dar. Die Mitarbeiter des Vertriebs, wie auch manchmal die Geschäftsführer selbst, halten zumeist Workshops beim Kunden Vorort ab. Nach

[1] Document Management System

einer darauffolgenden Spezifikationsphase die meist von den Geschäftsführern selbst abgehalten, erfolgt die Ticketerstellung im eigenen System. In der Designphase werden Demos je nach Leistungsmerkmalvielfalt als Feedbackschleife dem Kunden geliefert. In diesem Zusammenhang werden Mockups[1] mit Balsamic[2] entworfen um die Veranschaulichung und somit auch die Kundenbindung zu forcieren, und gleichzeitig das Risiko etwas zu minimieren [WK2013].

11.3.3 Firma HANS LOHR

Das Unternehmen[3] wurde 1979 in Form einer Kapitalgesellschaft mit beschränkter Haftung (GmbH) mit Unternehmenssitz in Wien gegründet. Als Unternehmensgegenstand befasst sich die Unternehmung vorwiegend mit dem Schaltschrankbau und Großprojekte im MSR[4]-Bereich. Ein weiteres Geschäftsfeld welches 1994 bereits eröffnet wurde, widmet sich generell der Entwicklung kleinerer Mechatroniken und Steuereinheiten wie sie im Bereich der Sicherheitstechnik und vor allem in der MSR Anwendung finden.

11.3.3.a Die Unternehmensstruktur

Die Unternehmung umfasst im Schnitt um die 120 Mitarbeiter. 10 Personen sind ausschließlich für den Vetrieb und die Kundenbetreuung zuständig, 15 weitere Mitarbeiter sind in der Administration, Verwaltung und Lagerlogistik tätig. Des Weiteren gibt es 5 Projektmanager und 5 Techniker die die Abteilung Entwicklung bilden. Für den Anlagenbau, die Montage und Wartung sind zirka weitere 85 Personen beschäftigt.

11.3.3.b Die Projektcharakteristik

Das Projekt, mit den internen Namen *LOHR-AlarmCenter,* ist eine mechatronische Komplettlösung für den Endanwender und Nutzer eines Alarmverbundsystems. Sein Haupteinsatzzweck findet in den Bereichen der Sicherheitstechnik Verwendung. Die Systemfamilie besteht aus vielen unterschiedlichen absetzbaren Sensoriken wie Rauch- Feuer und Bewegungsmelder. Die Anbindung erfolgt wahlweise über Draht oder drahtlosem Funk. Eine Zentraleinheit, die auch über eine Siemens kompatible Gebäudeleittechnik gemanagt werden soll, verwaltet die dezentralen Signalgeber welche meist als ODM[5]-Produkt erworben werden. Eine komfortable Bediensoftware konfiguriert alle Aspekte des Systems und protokolliert in non volatiler Weise auf Wunsch auch alle Ereignisfälle. Das System bedient sich proprietärer Techniken zur Verhinderung gewollten wie auch ungewollten Ausfalls.

Die derzeitige Konfigurationssoftware ist rein Windows basierend, es soll jedoch auch in Zukunft eine *App* für Mobiltelefone angeboten werden. Die Firmware der Zentraleinheit wird in C/C++ mit der opensource Toolchain *gcc*[6] programmiert.

[1] Wegwerfprototyp zur visuellen Veranschaulichung

[2] http://balsamiq.com/products/mockups/

[3] http://www.lohr.at/

[4] Mess- Steuer- und Regeltechnik

[5] Original Design Manufacturer, nicht zu verwechseln mit OEM

Die kundenseitige Konfigurationssoftware wird mit der klassischen .NET Technologie von MS-Windows realisiert, da hier einerseits bereits Erfahrungen gesammelt werden konnten, und andererseits die Interaktion der eingebetteten Entwicklungsumgebung in Windows ein hohes Maß an Stabilität und Support gewährleistet.

11.3.3.b.1 Die projektspezifische IT-Landschaft

In diesem Projekt kommt als Versionskontrollsystem Mercurial in der Version 2.7 zum Einsatz. Als Entwicklungsplattform wird Windows XP & Windows 7 eingesetzt. *MantisBT*[1] wird zur Fehlerverfolgung, rein intern und auch noch etwas spärlich verwendet. Entwicklungsinternes Wissen wird derzeit noch als einfacher Dateikatalog verwaltet, jedoch gibt es schon Bestrebungen die Wissensartefakte in das integrierte Wikisystem von MantisBT zu überführen. Die projektspezifische Kommunikation läuft strikt über Email.

11.3.3.c Die Projektdurchführung

Das Projekt läuft erst seit April 2014 und befindet sich pro Geräteinstanz in verschiedenen Reifestadien. Die Konfigurationssoftware ist derzeit in der Version 0.8, und wie die Versionsnummer verrät, ein noch offensichtlich junges Projekt, das bis dato noch keine Marktreife erlangt hat. Insgesamt gilt es als logischer Nachfolger der derzeit bestehenden und erfolgreich vermarkteten LOHR-Alarmsystemfamilie. Es wird zurzeit von 3 Entwicklern ständig bedient wobei der Abteilungsleiter zugleich die Funktion des Projektleiters übernimmt, sie stellen das eigentliche *Projektteam* dar.

Der Projektleiter erfasst die bereits intern wie auch am Markt identifizierten Mängel und bestehenden Kundenwünsche zu einem Projektanforderungskatalog zusammen. Dieser Gesamtkatalog wird im Vorfeld mit den Geschäftsführern diskutiert und abgewogen. Die effektiven Projektanforderungen werden dann mit den Empfehlungen des Vertriebes bezüglich Kosteneffizienz und sinnvollen Ergänzungen aus der Marktsicht harmonisiert und verabschiedet.

Der Projektleiter, sprich Abteilungsleiter, übernimmt dann mit einem ihm zugeteilten Geschäftsführer zusammen das vorerst rein intern ausgetragene Projekt. Sofern es erfolgreich realisiert und in Feldtests beziehungsweise in *friendly customer* Stellungen zufriedenstellend arbeitet, wird es als komplette Technologieablöse, um nicht einen *Kannibalismuseffekt* zu erzeugen, dem Markt zur Verfügung gestellt.

11.3.4 Firma KLINGER-GEBETSROITHER

Das Unternehmen[2] wurde 1918 in Form einer Personengesellschaft als Einzelunternehmung Max Gebetsroither mit Unternehmenssitz in Amstetten gegründet. 1975 wurde die Firma jedoch gekauft und in die Klinger-Gruppe integriert, diese Holding fungiert als Dachgesellschaft. Das Unternehmen selbst wurde bereits 1885

[6] https://gcc.gnu.org/

[1] https://www.mantisbt.org/

[2] http://www.klinger.co.at/

von Richard Klinger mit Sitz in Wien gegründet und besitzt zahlreiche Patente im Bereich Dichtungstechnik und Kugelhahnverschlusssysteme.

Als Unternehmensgegenstand befasst sich der Konzern vorwiegend mit der Herstellung von Großarmaturen und ist weltweit führend in Dichtungen. Er deckt zusätzlich auch eine große Bandreite möglicher Geschäftsbereiche des sekundären Zulieferbereiches ab. So sind auch Firmen im Bereich Kunststofffertigung, Maschinen- und Werkzeugbau sowie Kugellager und Steuerungen in der Unternehmensgruppe bereits seit längerem integriert.

11.3.4.a Die Unternehmensstruktur

Das Dachunternehmen, geführt als GmbH. & Co. KG., beschäftigt zirka 2.200 Personen verteilt auf über 60 Standorte weltweit. Der Stammsitz ist in Gumpoldskirchen, Österreich. Die komplexe Gruppenaufteilung erfolgt in divisionale wie auch funktionalen Bereichsebenen wie die folgende Veranschaulichung in Abbildung 11.2 verdeutlichen soll.

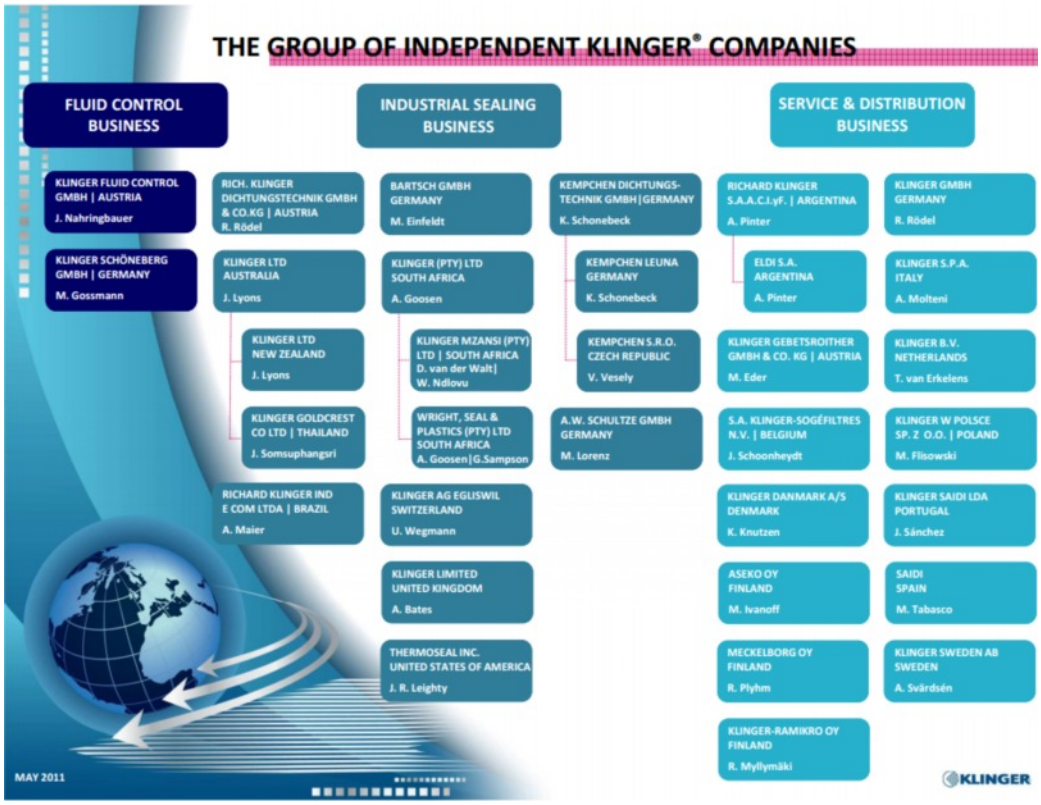


Abbildung 11.2 – die divisionale Aufteilung der Klinger Gruppe

11.3.4.b Die Projektcharakteristik

Das gegenständliche Projekt, das derzeit exklusiv am Stammsitz ausgetragen wird, ist betitelt mit den Namen *Klinger-Prüfstand. Es* ist eine Software für den Benutzer des firmeninternen Prüfstandes. Ihr Haupteinsatzzweck dient der computergestützten Durchführung und Protokollierung von Entwicklungstests an eigenen Stationen der Warmwasseraufbereitung sowie von Gerätschaften befreun-

deter Unternehmen. Um die eigene Qualitätssicherung reproduzierbar zu bewerkstelligen, wird sie auch als formal unterstützendes Werkzeug zur Durchführung stichprobenartiger Produktionstests genutzt. Das System verfügt über eine graphische Oberfläche wo diverse Parameter wie Differenzdrücke, Temperatursollwerte und Zapfmengen vorgegeben werden können. Nach dem Testfall kann die Auswertung mittels Diagrammen und auch in rein tabellarischer Form erfolgen. Die Generierung eines Druckreports mit wahlweiser Formatierung ist ebenfalls möglich und beansprucht den Charakter eines Referenzprotokolls.

Das standalone Programm ist eine monolithische Implementierung auf MS-Windows 7. Es besteht über die Hardwarekomponenten, welche analoge und digitale Einsteckkarten des Servers sind, eine starke Verzahnung und Plattformabhängigkeit, da auch die dazugehörigen Treiber-API's sehr *betriebssystemorientiert* sind. Das komplette System wurde rein mit der graphischen Sprache (gLang) des *National Instruments LabView 2013*[1] geschrieben und ist somit aus der Perspektive der reinen Applikationsschicht bedingt portabel.

11.3.4.b.1 Die projektspezifische IT-Landschaft

In diesem Projekt kommt Mercurial in der Version 3.0.2 zum Einsatz. Als Entwicklungsplattform wird ausschließlich Windows 7 eingesetzt. Das OpenSource Issue-Trackingsystem *Request-Tracker*[2] mit integriertem Ticketingsupport für Fehler und Feature-Requests befindet sich derzeit in der Evaluationsphase. Bis dato wurde eine proprietäre Datenbankapplikation zur einfachen Fehlerverfolgung benutzt, die bereits an die Grenzen ihrer Einsatzmöglichkeit stößt und auch kaum mehr gewartet wird. Unternehmensinternes Wissen wird mittels *Tagging Technik* semantisch geführter Dateibenennungen verwaltet, das dazu verwendete Werkzeug ist *TagStore*[3]. Eine firmeninterne Wissensuchmaschine & Expertensystem namens „Kloogle" ist in Planung. Die projektspezifische Kommunikation läuft derzeit hauptsächlich über Email und die MS-Office Kalender- und Teammeetingfunktionalitäten.

11.3.4.c Die Projektdurchführung

Das Projekt Klinger-Prüfstand läuft seit Jänner 2014 und befindet sich derzeit in der Version 3.1.0.0 Es wird von 1 Entwickler ständig weiterentwickelt, und einer weiteren Person die als Projektmanager fungiert, stellen das eigentliche *Projektteam* dar. Der Projektmanager spezifiziert die Kundenanforderungen direkt, wobei zu beachten ist, dass der Kunde selbst der Auftragnehmer ist. Es ist somit ein gänzlich firmeninternes Vorhaben. Die gewünschten Leistungsmerkmale werden mit den Bedarfsträgern der Testabteilungen und Qualitätssicherung diskutiert, vereinheitlicht und vereinbart. Das resultierende *Featureset* wird beschlossen und einem dedizierten Projektmanager zur Gesamtdurchführung übergeben. Hier findet sich die spezielle Form der in Kapitel 3.2.1.e vorgestellten „Reinen Projektorganisation" wieder.

[1] http://www.ni.com/labview/d/

[2] https://www.bestpractical.com/rt/issues.html

[3] http://tagstore.org/index.shtml

12 Kapitel

Fragestellungen und Manifestationen der Probanden

"Durch Testen kann man stets nur die Anwesenheit,
nie aber die Abwesenheit von Fehlern beweisen."
Edsger Dijkstra

Schwerpunkte in diesem Kapitel:

- Die Evaluationsprozedur
- Die Strategie der Fragestellungen
- Die Manifeste

12.1 Die Evaluationsprozedur

Um die aufgestellten Hypothesen im praktischen Umfeld der Privatwirtschaft auch sinnvoll verifizieren zu können bedarf es mehrerer Faktoren. Vorerstes muss, abgesehen von den vorangegangenen kollaborativen Erhebungen, auch eine weiterführende Kooperationsbereitschaft mit den etablierten Industriepartnern geschaffen werden. Anschließend werden passende Erhebungsverfahren angestrebt um die Hypothesen seriös bewerten zu können.

Wie schon in Kapitel 11.1 erwähnt, werden hierbei eine (standardisierte) quantitative Fragebogenerhebung wie im Anhang in den Abbildungen 15.1 & 15.2 zu sehen und im Anschluss daran eine selektive (nicht standardisierte) qualitative Interviewbefragung wie in Abbildung 15.3 & 15.4 durchgeführt. Diese beiden Betrachtungsweisen, ergänzt mit der subjektiv erworbenen Erkenntnis aus der Feldforschung Vorort, ergeben somit ein holistisches nachvollziehbares Bild woraus pragmatische Schlüsse mit Rückschluss auf eine wissenschaftliche Verallgemeinerung gewonnen werden können. Diese gewonnen Erkenntnisse können dazu genutzt werden, um in einer nächsten Iterationsphase des Forschungsprojektes, richtungsweisende beziehungsweise spezifischerer Vorgaben zu stellen.

Es wurden auch Anstrengungen unternommen diese Aussagekraft zu erhöhen, in dem vermehrt Firmen mit heterogener Projektstruktur als Probanden dienten. So sind reine Softwareunternehmen genauso wie mechatronische Firmen involviert. Die Unternehmensgröße reicht von acht lokal beschäftigten Personen bis hin zu theoretisch 2200 Mitarbeitern weltweit wobei, hier noch wichtiger, die Unternehmensstruktur im Vordergrund steht.

12.2 Die Strategie der Fragestellungen

Der gesamte Fragenkatalog zielt prinzipiell durch direkte oder indirekte Formulierung auf die acht postulierten Hypothesen ab. Es gibt auch Nebenfragen die allgemeinere Fragestellungen mit beantworten sollen. Die Überlegungen zu den erdachten Fragen sollen im Folgenden kurz porträtiert werden.

Es wurde zuerst bewusst zwischen Managern und Technikern unterschieden, deswegen, da jene Benutzergruppen auch jeweils unterschiedlich involviert sind. Die Techniker entsprechen sinngemäß den Systembenutzern, den Operatoren, sie bedienen es vorwiegend. Den Nutzen daraus ziehen mehrheitlich die Manager, sie stellen somit quasi die Nutzerschaft dar. Dem entsprechend ist auch der Fokus der Frageformulierung anders gestellt.

Es gibt zwar überschneidende Fragekategorien die beide betreffen, jedoch füllt der Rest des jeweiligen Fragekataloges jeweils die komplementären Lücken. Den Technikern werden vermehrt systemtechnische und prozessrelevante Fragen gestellt, den Managern hingegen vorwiegend Fragen gemäß Usability und jeweiligem Interpretationsgehalt.

Ein weiteres Augenmerk galt der Formulierung. Es wurden bewusst die meisten Fragen so gestaltet, dass kein subjektiver Trend vermittelt wurde um nicht den Probanden unnötigerweise zu Vorurteilen zu verleiten. Bei den Interviewfragen wurde verstärkt der Versuch unternommen rein persönliche Empfindungen und subjektive Einsichten, die nur mittels eines persönlichen Gespräches erörtert werden können, festzuhalten – deswegen wurden sich auch nicht rein numerisch bewertet sondern vielmehr subjektiv gewichtet. Manche Fragen stellen, in den quantitativ zu erfassenden Fragekatalog sowie in den Interviews, dieselbe Frage nochmals, sind jedoch bewusst anders formuliert um auch etwaige unterschiedliche Interpretationen zu erfassen.

12.3 Die Manifeste

Die auf den nächsten Seiten folgenden eingescannten Niederschriften manifestieren die jeweiligen exerzierten Erprobungsphasen pro Unternehmen wobei jedes Dokument mit dem firmeneigenen Stempel samt Unterschrift des jeweiligen Geschäftsführers unterfertigt ist. Mit diesem Akt wird versucht trotz der unternehmensweiten anonymisierten Behandlung der projektspezifischen Daten sowie der Beteiligten samt ihrer Stellungnahmen einen verbindlichen sowie integren Abschluss zu bilden.

Es werden vor allem alle wesentlichen Eckdaten des Evaluationszeitraumes dokumentiert. Dies wird erreicht, indem das geforderte Spektrum sinnvoll durch die bekannten 5W-Fragen abgedeckt wird. *Wer, Was, Wann, Wo und Wie, w*obei das Wie die Erhebung der Daten mittels Fragebogentechnik und Interviews darstellt, und das Wer sich ausschließlich auf das Unternehmen an sich bezieht.

MANIFEST DER FA. LEGIOKILL

Die Geschäftsführung der Firma LEGIOKILL GmbH. bestätigt hiermit, dass das im Rahmen der Dissertationarbeit entwickelte Software-Werkzeug RBMST von Herrn Ing. Mag. Marcello Presulli in unserem Unternehmen im vereinbarten Testzeitraum von 01.09.2014 bis 30.09.2014 am Projekt Legio-Manager praxisnahe exerziert wurde.

Innerhalb dieser Testperiode wurden insgesamt ca. 8 *Projektsnapshots* gemacht und einige davon wurden in formellen wie auch informellen Besprechungen zur Informationsunterstützung herangezogen. Die Feststellung der Nutzbarkeit wurde am 01.10.2014 an unseren Projektmitarbeitern mittels Fragebogenerhebung und Interviewtechniken in unserem Hause anonym durchgeführt. Die Interviews wurden nach Erfassung und Kontrolle von den jeweiligen Befragten mit den Unternehmensstempel firmiert.

Insgesamt hat sich das Werkzeug als hilfreich erwiesen. Mögliche Verbesserungsvorschläge wurden unsererseits bereits kommuniziert. Der Einsatz dieser Software hat als Nebeneffekt einige positive Änderungen in unserem Arbeitsablauf forciert. Ein Weiterverfolgen des Konzepts und stabilisieren der Softwarefunktionaltäten wird aus unternehmerischer Sicht als sinnvoll erachtet.

Das Unternehmen hat maßgebliche Beiträge in den folgenden kollaborativen Diskurs-Sitzungen geleistet:

1. *Sitzung am 15.12.2011: Sinnhaftigkeit, Nutzbarkeit und Abgrenzungen*
2. *Sitzung am 30.01.2012: Ontologien, Domänenspezifische Sprache*
3. *Sitzung am 02.05.2012: Schlüsselwörter, Benutzerschnittstellen*
4. *Sitzung am 25.10.2012: Schlüsselwörter, Ergonomie, Visualisierung*
5. *Sitzung am 14.03.2013: Schlüsselwortpragmatismus, Modifizierbarkeit*
6. *Sitzung am 20.06.2013: Verschlüsselung, Schnittstellen zu Fremdsystemen*

Dieses Manifest wurde gemeinsam und sinngemäß seitens der Firma LEGIOKILL GmbH. mit dem Autor der Dissertation verfasst und niedergeschrieben.

Datum 01.10.2014 *firmenmäßige Zeichnung und Unterschrift*

LEGIOKILL
HERNALSER
HAUPTSTRASSE 169
A-1170 WIEN
TEL.: +43 1 479 31 17
office@legiokill.at www.legiokill.at

Gerald Kures

Ing. Mag. Marcello Presulli 01.10.2014

Verzeichnisgestützte Managementsysteme zur kollaborativen Projektunterstützung in verteilten Software Entwicklungen

MANIFEST DER FA. METAMAGIX

Die Geschäftsführung der Firma METAMAGIX GmbH. bestätigt hiermit, dass das im Rahmen der Dissertationarbeit entwickelte Software-Werkzeug RBMST von Herrn Ing. Mag. Marcello Presulli in unserem Unternehmen im vereinbarten Testzeitraum von 01.09.2014 bis 30.09.2014 am Projekt NQZ praxisnahe exerziert wurde.

Innerhalb dieser Testperiode wurden insgesamt ca. 10 *Projektsnapshots* gemacht und einige davon wurden in formellen wie auch informellen Besprechungen zur Informationsunterstützung herangezogen. Die Feststellung der Nutzbarkeit wurde am 02.10.2014 an unseren Projektmitarbeitern mittels Fragebogenerhebung und Interviewtechniken in unserem Hause anonym durchgeführt. Die Interviews wurden nach Erfassung und Kontrolle von den jeweiligen Befragten mit den Unternehmensstempel firmiert.

Insgesamt hat sich das Werkzeug als hilfreich erwiesen. Mögliche Verbesserungsvorschläge wurden unsererseits bereits kommuniziert. Ein Weiterverfolgen des Konzeptes und stabilisieren der Softwarefunktionaltäten wird aus unternehmerischer Sicht angeraten. Ein besonderes Augenmerk sollte hier vorallem im Bereich Software-Ergonomie und Prozessautomatisierung gewidmet werden.

Das Unternehmen hat maßgebliche Beiträge in den folgenden kollaborativen Diskurs-Sitzungen geleistet:

1. *Sitzung am 25.10.2012: Schlüsselwörter, Ergonomie, Visualisierung*
2. *Sitzung am 14.03.2013: Schlüsselwortpragmatismus, Modifizierbarkeit*
3. *Sitzung am 20.06.2013: Verschlüsselung, Schnittstellen zu Fremdsystemen*

Dieses Manifest wurde gemeinsam und sinngemäß seitens der Firma METAMAGIX GmbH. mit dem Autor der Dissertation verfasst und niedergeschrieben.

Datum
02.10.2014

firmenmässige Zeichnung und Unterschrift

metamagix
Software & Consulting GmbH
Kienmayergasse 16
1140 Wien, Österreich
T: +43 1 9902804
www.metamagix.net

DI Gilly & DI Kepplinger

Ing. Mag. Marcello Presulli

02.10.2014

MANIFEST DER FA. HANS LOHR

Die Geschäftsführung der Firma Hans LOHR GmbH. bestätigt hiermit, dass das im Rahmen der Dissertationarbeit entwickelte Software-Werkzeug RBMST von Herrn Ing. Mag. Marcello Presulli in unserem Unternehmen im vereinbarten Testzeitraum von 01.09.2014 bis 30.09.2014 am Projekt Lohr-Alarmcenter praxisnahe exerziert wurde.

Innerhalb dieser Testperiode wurden insgesamt ca. 6 *Projektsnapshots* gemacht und einige davon wurden in formellen wie auch informellen Besprechungen zur Informationsunterstützung herangezogen. Die Feststellung der Nutzbarkeit wurde am 03.10.2014 an unseren Projektmitarbeitern mittels Fragebogenerhebung und Interviewtechniken in unserem Hause anonym durchgeführt. Die Interviews wurden nach Erfassung und Kontrolle von den jeweiligen Befragten mit den Unternehmensstempel firmiert.

Insgesamt hat sich das Werkzeug als hilfreich erwiesen. Mögliche Verbesserungsvorschläge wurden unsererseits bereits kommuniziert. Der Einsatz dieser Software hat einige positiven Änderungen im Bereich unseres Wissensmanagements erwirkt. Ein Weiterverfolgen des Konzeptes und stabilisieren der Softwarefunktionaltäten wird aus unternehmerischer Sicht empfohlen. Des Weiteren sollte das Werkzeug speziell im Bereich der Metriken erweitert werden.

Das Unternehmen hat maßgebliche Beiträge in den folgenden kollaborativen Diskurs-Sitzungen geleistet:

1. *Sitzung am 15.12.2011: Sinnhaftigkeit, Nützlichkeit und Abgrenzungen*
2. *Sitzung am 02.05.2012: Schlüsselwörter, Benutzerschnittstellen*
3. *Sitzung am 25.10.2012: Schlüsselwörter, Ergonomie, Visualisierung*
4. *Sitzung am 14.03.2013: Schlüsselwortpragmatismus, Modifizierbarkeit*
5. *Sitzung am 20.06.2013: Verschlüsselung, Schnittstellen zu Fremdsystemen*

Dieses Manifest wurde gemeinsam und sinngemäß seitens der Fa. Hans LOHR GmbH. mit dem Autor der Dissertation verfasst und niedergeschrieben.

Datum | *firmenmässige Zeichnung und Unterschrift*

03.10.2014

HANS LOHR Gesellschaft m.b.H.
A-1230 Wien, ...
tel +43-1-9859585-0 fax DW 633

KommRat. Günter Slabihoud

Ing. Mag. Marcello Presulli | 03.10.2014

MANIFEST DER FA. KLINGER-GEBETSROITHER

Die Firma KLINGER-GEBETSROITHER GmbH. bestätigt hiermit, dass das im Rahmen der Dissertationarbeit entwickelte Software-Werkzeug RBMST von Herrn Ing. Mag. Marcello Presulli in unserem Unternehmen im vereinbarten Testzeitraum von 01.09.2014 bis 30.09.2014 am Projekt Legio-Manager praxisnahe exerziert wurde.

Innerhalb dieser Testperiode wurden insgesamt ca. 5 *Projektsnapshots* gemacht und einige davon wurden in formellen wie auch informellen Besprechungen zur Informationsunterstützung herangezogen. Die Feststellung der Nutzbarkeit wurde am 01.10.2014 an unseren Projektmitarbeitern mittels Fragebogenerhebung und Interviewtechniken in unserem Hause anonym durchgeführt. Die Interviews wurden nach Erfassung und Kontrolle von den jeweiligen Befragten mit den Unternehmensstempel firmiert.

Insgesamt konnte sich keine direkte Nutzbarkeit des Werkzeug erweisen. Es konnten keine effektiven Verbesserungen hinsichtlich des Projektmanagements erzielt werden, jedoch wurden generelle Techniken zur verbesserten Projektaufbereitung vakant. Es wird in diesem Zusammenhang auch auf die spezielle Konstellation des Projektes hingewiesen. Ein Weiterverfolgen des Konzeptes wird aus der unternehmerischen Sicht dennoch als sinnvoll erachtet, da es bei klassischen Projektorganisationen durchaus Potential hat.

Das Unternehmen hat maßgebliche Beiträge in den folgenden kollaborativen Diskurs-Sitzungen geleistet:

1. *Sitzung am 25.10.2012: Schlüsselwörter, Ergonomie, Visualisierung*
2. *Sitzung am 14.03.2013: Schlüsselwortpragmatismus, Modifizierbarkeit*
3. *Sitzung am 20.06.2013: Verschlüsselung, Schnittstellen zu Fremdsystemen*
4. *Sitzung am 7.02.2014: theoretische internationale Einsetzbarkeit & Skaleneffekte*

Dieses Manifest wurde gemeinsam und sinngemäß seitens der verantwortlichen Projektleitung der Fa. KLINGER-GEBETSROITHER GmbH. mit dem Autor der Dissertation verfasst und niedergeschrieben.

Datum — *firmenmässige Zeichnung und Unterschrift*

01.10.2014

TEL. 02252/60 71 00-0
FAX 02252/60 71 00-3010
KLINGER Gebetsroither GmbH & Co KG
Ing. Roland Bezenker
Am Kanal 8-10
A-2352 Gumpoldskirchen

Ing. Mag. Marcello Presulli — 01.10.2014

13 Kapitel

Die Auswertungsverfahren und ihre Ergebnisse

"Unser Wissen ist ein kritisches Raten,
ein Netz von Hypothesen,
ein Gewebe von Vermutungen."
Karl Popper

Schwerpunkte in diesem Kapitel:

- Die Auswertung
- Die Rohdaten
- Die Hypothesenbewertung
- Schlussfolgerungen

13.1 Die Auswertung

13.1.1 Methodentriangulation

Wie in der empirischen Sozialforschung üblich, dienen mehrdimensionale Erhebungsverfahren einer differenzierten Betrachtungsweise. Es wird dabei der Versuch unternommen qualitativ & quantitativ ermittelte Ergebnisse zu kombinieren um ein vollständigeres Gesamtbild zu erhalten. Durch diese Hybridisierung wird der Forscher in die Lage versetzt, auch subtilere Erscheinungsformen möglicherweise zu erfassen, mit dem Ziel insgesamt eine höhere Gültigkeit der Aussagen zu erhalten. Wichtig dabei ist, eine optimale Gewichtung der einzelnen Verfahren, da nicht jede Analyse in der gleichen Intensität und Zielformulierung stattfinden kann. Durch die unterschiedlichen Bewertungsverfahren erhöht sich auch die Wahrscheinlichkeit, dass sich dadurch gewisse Fehler und Messungenauigkeiten kompensieren. Prinzipiell besitzt die Fragebogenauswertung, aufgrund der Stichprobengröße, die höchste Gewichtung.

Zusätzlich mit Hilfe einer dissoziierten Interpretationsgewichtung von J*a, Nein und Indifferent* der korrespondierenden Interviewantworten kann so individuell die Bewertung jeder einzelnen Hypothese vorgenommen werden. Bei indifferenten Resultaten werden gegebenenfalls die Beobachtungen aus der Feldforschung herangezogen um zumindest gewisse Tendenzen abzuleiten zu können. In einer möglicherweise nächsten Iterationsphase der Erprobung können durch das weiterentwickelte Erhebungsverfahren jene Fragen gegebenenfalls effektiver erörtert

werden. Die Zusammenfassung der quantitativen sowie qualitativen Einzelergebnisse befindet sich unter 13.2.6 & 13.2.7. Gewonnene Erkenntnisse aus der partizipativen Feldforschung werden unter 13.4.2 konzeptionell diskutiert.

13.1.1.a Quantitative Analyse

Dieses standardisierte Erhebungsverfahren dient der breiten Erfassung relevanter Querschnittsdaten. Als Verfahrensmittel wurde eine völlig anonyme elektronische Befragung durchgeführt. Der Ablauf entspricht, wie in Abbildung 11.1 erkennbar, dem Modus „Bildschirm-Interview“ der dazu verwendeten Software *Grafstat* und stellt somit ein nachvollziehbares Vehikel der quantitativen Analyse dar. Der Vorteil ist hierbei, die homogene Erfassung der Antworten und die formale Darstellungs- und Evaluationsmethodik. In diesem Verfahren wurden jeweils alle Probanden befragt.

Dazu zählen im Kern das Projektteam, wie es in Kapitel 11.4 pro Unternehmen vorgestellt wurde, und gegebenenfalls auch die bei dem Projektstatusmeeting anwesenden Entscheidungsträger. Dem entsprechend ist auch die Anzahl der Instanzen unterschiedlich groß, wie man an den Zählern in den statistischen Aufschlüsselungen erkennen kann.

Da die Hypothesen auch auf die unterschiedlichen Einsatzdomänen abgebildet werden, variieren je nach Ausrichtung, technisch oder verwaltungstechnisch, die Fragenstellungen gemäß dem Einsatzschwerpunkt von RBMST. Des Weiteren gibt es auch einen Fragenkomplex der allgemeinem Interesse dient oder das Ziel hat, den Implementationsstand selbst zu erkunden, um eine weiterführende Feature Roadmap für RBMST entwerfen zu können [HI2014]. Diese Kategorie der Fragestellungen wird meistens sehr einfach in Form von Entscheidungsfragen abgehandelt, um nur eine prinzipielle Neigung zu erfassen.

13.1.1.a.1 Angewandte statistische Methoden

Grafstat ermöglich eine automatische Grundauswertung der erhobenen Daten. Es werden dazu einfache statistische Lagemaße zur Bewertung herangezogen. So wird, abhängig von der konkreten Fragenwertung, sicherlich ein Minimum Maximum Intervall angegeben, der arithmetische Mittelwert errechnet sowie gegebenenfalls auch der zugehörige Median ermittelt. Die jeweiligen Zähler werden auf die Grundgesamtheit in Prozenten ausgedrückt. Es werden auch jene Fragen wo keine Antwort oder auch eine Mehrfachauswahl möglich war berücksichtigt und korrekt behandelt.

Darauf basierend bieten sich weitere Möglichkeiten zur logischen Filterung mittels AND, OR, XOR und NOT an. Eine selektive Auswahl erlaubt eine Kopplung und Gruppierung ähnlicher Fragestellungen zu einer Aussage beziehungsweise über den Kreuztabellenmechanismus bietet sich auch die Möglichkeit einer zweidimensionalen Auswertung nach vorhandenen Kriterien an. Die folgende Analyse setzt

auf der Grunddatenauswertung auf und bewertet mittels einer aggregierten Kopplung statistischer Ergebnisse die postulierten Hypothesen.

13.1.1.b Qualitative Analyse

In der Literatur wird sie als eine nicht standardisierte Ermittlung von Fakten durch interpretative, sogar hermeneutische Methoden definiert. Hier fungiert sie als ein wesentliches Instrument der Analyse, um diverse Verhalten und Neigungen in Bezug auf die Einsatzfähigkeit von RBMST zu ermitteln. Wie schon eingangs erwähnt wird sie wie der elektronische Fragebogen völlig anonym aber im Gegensatz dazu selektiv durchgeführt. Die Selektion wurde primär auf Basis der Beobachtungen aus der Feldforschung durchgeführt.

Pro Projektverlauf, wurden ein bis zwei *Schlüsselfiguren* innerhalb der Unternehmen identifiziert und dementsprechend interviewt. Eine Schlüsselfigur bedeutet in diesem Kontext, eine Person mit hoher Schnittstellen- und/oder Entscheidungsfunktion auf Projektebene. Auch hier unterschieden sich die hypothesenzentrierten Fragen bezogen auf eine technische oder managementorientierte Ausrichtung mit teilweise auch rein implementatorischen Fragen. Wie in Abbildung 13.1 & 13.2 beispielhaft gezeigt, wird der stichwortartig ausgefüllte Interviewbogen jedoch abschließend von der jeweilig befragten Person auf die Richtigkeit der Aussagen hin überprüft und anschließend mit dem unternehmenseigenen Stempel bestätigt.

13.1.1.c Erfahrungen aus der Feldforschung

Eine essentielle Ergänzung der beiden oben aufgezählten Analyseverfahren stellt die gewonnene Erfahrung aus der Feldforschung dar. Diese Phase der Erfahrungsanreicherung verläuft sehr punktuell und intensiv. Die explizite Aussagekraft der Erfahrungswerte ist gering, da sie implizit mit den formulierten Hypothesen verwoben sind und dementsprechend nicht klar angegrenzt werden können.

Es sind auch diffizile Beobachtungen die sich auch nur im Gruppenkontext beziehungsweise im Unternehmensverbund zeigen und auch deswegen schwer explizierbar sind, dennoch für ein ganzheitliches Verständnis unerlässlich. Nichts desto trotz wird versucht nachfolgend punktuell jene Erfahrungswerte konzeptionell zu diskutieren. Wie sich noch unter 13.3.10.b zeigen wird, dass nicht alle Beobachtungen stets positiver Natur sind.

13.2 Die Rohdaten

13.2.1 Die konsolidierte Grundauswertung

Die folgende Statistik beruht, wie in den Tabellen 13.1 & 13.2 dargestellt, auf der aggregierten Zusammenstellung aller Einzelergebnisse der jeweiligen 4 Unternehmen. Die Grundauswertung der Fragebögen erfolgt vollautomatisch und generiert ein *RichText* formatiertes Dokument. Basierend auf diesen deskriptiven Rohdaten wurden anschließend die Zählerwahrscheinlichkeiten der zugehörigen Fragen miteinander gekoppelt.

13.2.2 Management zentrierte Fragebogenauswertung

Grundauswertung Teil 1 der Befragung:

rbmst-manager

1) Wie viele projektrelevante Meetings mit Entscheidungscharakter wurden in der Erprobungsphase abgehalten ?

Antworten	8
ohne Antwort	0
Minimum	3
Maximum	5
Mittelwert	3,875

2) Wie viele Artefaktausgaben wurden während der Erprobungsphase erzeugt ?

Antworten	8
ohne Antwort	0
Minimum	2
Maximum	5
Mittelwert	3,375

3) Wie oft wurden dabei die graphische Artefaktausgabe herangezogen ?

Antworten	8
ohne Antwort	0
Minimum	1
Maximum	5
Mittelwert	3

4) Hat sich eine bedingt erhöhte Informationsverfügbarkeit eingestellt ?

sehr	0	(0,00%)
eher	5	(62,50%)
wenig	2	(25,00%)
nicht	1	(12,50%)
Summe	8	
ohne Antwort	0	

5) Wie oft war während der Informationsbeschaffung das betreffende Repository nicht verfügbar, wenn nicht warum ?

0	6	(75,00%)
<=3	0	(0,00%)
<=5	0	(0,00%)
>5	2	(25,00%)
Summe	8	
ohne Antwort	0	

<4/5> HG-WebServer vergessen zu aktivieren
<7/8> selbst schuld, lokales HG-WebService war down

6) Wie oft wurde die kryptographische Übermittlung verwendet, wenn ja für welchen Zweck (Vertraulichkeit, Authentifikation) ?

Nie	4	(50,00%)
<=5	0	(0,00%)
<=10	0	(0,00%)

>10	4	(50,00%)
Summe	8	
ohne Antwort	0	

<4/5> zum Testen der Authentifikation
<5/6> als Test
<6/7> als vertrauliche Übermittlung an Partnerunternehmen
<7/8> als mögliches Instrument zur Projektübersicht mit JointVenture Unternehmung

7) Hat sich die bedarfsorientierte Informationsintegrität durch das Pseudowasserzeichen erhöht ?

sehr	1	(12,50%)
eher	3	(37,50%)
wenig	3	(37,50%)
nicht	1	(12,50%)
Summe	8	
ohne Antwort	0	

8) Wurde der bedarfsorientierte Informationsaustausch bzw. Wissensaustausch durch den Einsatz des Systems forciert ?

sehr	1	(12,50%)
eher	4	(50,00%)
wenig	3	(37,50%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	2,25	
Median	2	

9) Dienen die Darstellungen einer verbesserten Repräsentation der Information und des Projektzustandes ?

sehr	5	(62,50%)
eher	3	(37,50%)
wenig	0	(0,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	1,38	
Median	1	

10) Ist die Aussagekraft eines Projektzustandes bezüglich der erhöhten Informationstransparenz verbessert ?

sehr	0	(0,00%)
eher	6	(75,00%)
wenig	2	(25,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	

Mittelwert	2,25	
Median	2	

11) Deckt die Anzahl der vereinbarten Schlüsselwörter, und somit die Ausdrucksform, die Bedarfsfälle ausreichend ab ?

sehr	0	(0,00%)
eher	3	(37,50%)
wenig	4	(50,00%)
nicht	1	(12,50%)
Summe	8	
ohne Antwort	0	
Mittelwert	2,75	
Median	3	

12) Ergibt sich durch die Nutzung der Repository-Metadaten ein besseres Gefühl der Sicherheit aufgrund der originären Informationsquelle ?

sehr	1	(12,50%)
eher	7	(87,50%)
wenig	0	(0,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	

13) Wurde Zeit bei der Beschaffung relevanter Informationen im Vergleich zur ursprünglichen Erfassungsmethode eingespart ?

sehr	1	(12,50%)
eher	4	(50,00%)
wenig	2	(25,00%)
nicht	1	(12,50%)
Summe	8	
ohne Antwort	0	
Mittelwert	2,38	
Median	2	

14) Ist die Systemverarbeitungszeit, unabhängig von der Repositoryanbindung, für eine adäquate Informationsbeschaffung ausreichend ?

sehr	7	(87,50%)
eher	1	(12,50%)
wenig	0	(0,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	1,13	
Median	1	

15) Wird das Sicherheitsgefühl bei Entscheidungsfindungen basierend auf den Ergebnissen des Systems bestärkt ?

sehr	0	(0,00%)
eher	6	(75,00%)

wenig	2	(25,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	2,25	
Median	2	

16) Wurde durch die automatische Erhebung gleichgearteter Informationsbedürfnisse die Ermittlungsphase zeitlich entlastet ?

sehr	3	(37,50%)
eher	4	(50,00%)
wenig	1	(12,50%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	

17) Hat die Anzahl der tatsächlich benötigten Informationsträger abgenommen ?

sehr	0	(0,00%)
eher	1	(12,50%)
wenig	2	(25,00%)
nicht	5	(62,50%)
Summe	8	
ohne Antwort	0	

18) Hat sich insgesamt ein kollektiv verbessertes Projektverständnis eingestellt ?

sehr	0	(0,00%)
eher	4	(50,00%)
wenig	4	(50,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	

19) Welche Kommunikationsform wurde zur Beantwortung von Anfragen (enquiry) verwendet, wenn andere, welche ?

email	4	(50,00%)
verbal	2	(25,00%)
andere	2	(25,00%)
Summe	8	
ohne Antwort	0	

<6/7> RedMine Ticketing
<7/8> Tickets

20) Zeichnen sich durch die effizientere Informationsermittlung auch künftige Projektengpässe besser ab ?

ja	6	(75,00%)
nein	2	(25,00%)
Summe	8	

ohne Antwort	0	

21) Ist der Detail- und Navigationsgrad der graphischen Repräsentation ausreichend, wenn nein warum ?

ja	5	(62,50%)
nein	3	(37,50%)
Summe	8	
ohne Antwort	0	

<4/5> bessere frabliche Zuordnung
<6/7> einfachere Zuordnung der Knoten erwünscht
<8/9> mehr farbunterschiede sinnvoll

22) Hat sich durch die formale Aufbereitung der Information ein konsensualer gruppenübergreifender Projektdiskurs ergeben, wenn ja hauptsächlich wodurch ?

nein	5	(62,50%)
ja	3	(37,50%)
Summe	8	
ohne Antwort	0	

<5/6> durch die Darstellung sind einige Abhängigkeiten aufgefallen die dann weiter besprochen wurden
<6/7> eher technischerweise, da über den Einsatz einer ähnlichen Technologie diskutiert wird
<8/9> die interessante Idee der technischen Lösung wurde intern besprochen

23) Wenn die Artefaktausgabe auch als Entscheidungsgrundlage fungiert hat, welcher Metrikfaktor wurde dabei verwendet ?

nein	5	(62,50%)
ja	3	(37,50%)
Summe	8	
ohne Antwort	0	

<4/5> Stundensatz
<5/6> Lizenzkostenrechnung
<6/7> Einheitswert 1 um den echten Sekundenwert zu ermitteln, der für ein anderes Werkzeug genutzt wird

Tabelle 13.1 – Die Grundauswertung des RBMST-Manager Fragebogens

13.2.3 Technisch orientierte Fragebogenauswertung

Grundauswertung Teil 1 der Befragung:

rbmst-techniker

1) Wie oft wurde eine konventionelle Konsistenzanalyse der injektierten Schlüsselwortpragmatiken vollzogen ?

Antworten	8
ohne Antwort	0
Minimum	0
Maximum	1
Mittelwert	0,5

2) Wie viele Fehleingaben bei den Schlüsselwortpragmatiken wurden selbstständig bemerkt und auch korrigiert ?

0	3	(37,50%)
<=2	5	(62,50%)
<=5	1	(12,50%)
>5	0	(0,00%)
Summe	8	
ohne Antwort	0	

3) Welches Schlüsselwort wurden bei der Metadateninjektion am häufigsten verwendet ?

state	4	(50,00%)
dependency	3	(37,50%)
enquiry	0	(0,00%)
knowledgeSource	1	(12,50%)
Summe	8	
ohne Antwort	0	

4) Wie lange dauert durchschnittlich die exklusive Eingabe einer Schlüsselwortpragmatik, wenn >90 Sekunden warum ?

<=90 Sekunden	6	(75,00%)
>90 Sekunden	2	(25,00%)
Summe	8	
ohne Antwort	0	

<5/6> wenn die Verlaufsfunktion nicht genutzt wird, dann kommt es auf die Information an
<8/9> kommt auf das Informationsargument an

5) Sind die voreingestellten regulären Ausdrücke der Filter (\w+) für eine Informationsabbildung und nachträgliche Informationsextraktion ausreichend, wenn nicht welcher reguläre Ausdruck wäre adäquat ?

sehr	0	(0,00%)
eher	4	(50,00%)
wenig	2	(25,00%)
nicht	2	(25,00%)
Summe	8	
ohne Antwort	0	

<6/7> es soll nicht nur 0-9,A-Z und _ erlaubt sein, sollte auch HTTP-Link funktionieren

<8/9> besser etwas mehr offener, z.B. für URLs

6) Ist die Notationsform der Schlüsselwortpragmatik (Syntax) intuitiv anwendbar ?

sehr	0	(0,00%)
eher	6	(75,00%)
wenig	2	(25,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	2,25	
Median	2	

7) Wie aufwändig ist die Metadateninjektion im Bezug auf die subjektive Nutzbarkeit ?

sehr	0	(0,00%)
eher	5	(62,50%)
wenig	3	(37,50%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	2,38	
Median	2	

8) Hat sich ein besseres semantisches Verständnis zwischen den Projektbeteiligten etablieren können ?

sehr	0	(0,00%)
eher	6	(75,00%)
wenig	2	(25,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	2,25	
Median	2	

9) Ist die Auflösung der Arbeitspakete durch die Untergliederung der Punktnotation hinreichend ?

sehr	1	(12,50%)
eher	4	(50,00%)
wenig	3	(37,50%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	2,25	
Median	2	

10) Ergibt sich durch die Nutzung der Repository-Metadaten eine besseres Gefühl

hinsichtlich der Informationstransparenz bei Projektentscheidungen ?

sehr	3	(37,50%)
eher	5	(62,50%)
wenig	0	(0,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	1,63	
Median	2	

11) Deckt die Anzahl der vereinbarten Schlüsselwörter, und somit die Ausdrucksform, die Bedarfsfälle ausreichend ab ?

sehr	0	(0,00%)
eher	4	(50,00%)
wenig	4	(50,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	2,5	
Median	2	

12) Hat sich durch die Nutzung der Ausgabeartefakte auch die projektbezogene Informationsrepräsentation in den jeweiligen Meetings verbessert ?

sehr	0	(0,00%)
eher	8	(100,00%)
wenig	0	(0,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	2	
Median	2	

13) Wäre ein explizites Werkzeug zur geführten Metadateninjektion, welche alle systemischen Anforderungen abdeckt, sinnvoll ?

sehr	4	(50,00%)
eher	4	(50,00%)
wenig	0	(0,00%)
nicht	0	(0,00%)
Summe	8	
ohne Antwort	0	
Mittelwert	1,5	
Median	1	

14) Hat sich durch die Möglichkeit der Metadateninjektion während des Projektverlaufes die Situation bezüglich einer ad hoc Informationsbereitstellung entspannt, wenn ja inwiefern ?

sehr	0	(0,00%)
eher	0	(0,00%)
wenig	2	(25,00%)
nicht	6	(75,00%)

Summe	8
ohne Antwort	0

<4/5> es können Projektabhängigkeiten klarer dargestellt werden

<7/8> maximal die Projektabhängigkeiten

15) Ist eine nachträgliche Modifikation bereits injektierter Schlüsselwortpragmatiken komplex ?

ja	6	(75,00%)
nein	2	(25,00%)
Summe	8	
ohne Antwort	0	

16) Wird die Eingabe sofort nach erfolgreichem commit bewerkstelligt ?

ja	5	(62,50%)
nein	3	(37,50%)
Summe	8	
ohne Antwort	0	

17) Wird die Schlüsselwortpragmatik vom Autor des commits selbst erstellt ?

ja	5	(62,50%)
nein	3	(37,50%)
Summe	8	
ohne Antwort	0	

18) Ist die Eingabe der Lesezeichen über die graphische Oberfläche TortoiseHG komfortabel ?

ja	7	(87,50%)
nein	1	(12,50%)
Summe	8	
ohne Antwort	0	

19) Wird der RBMST-Systemeinsatz oft nur an die Releaseversionen gekoppelt ?

nein	6	(75,00%)
ja	2	(25,00%)
Summe	8	
ohne Antwort	0	

20) Wird die Darstellung des Datums und der Kontaktdaten des Autors in der Artefaktausgabe als unangenehm erachtet ?

ja	4	(50,00%)
nein	4	(50,00%)
Summe	8	
ohne Antwort	0	

21) Wäre eine Definition einer eigenständigen Ontologie (Begriffswelt) von Vorteil ?

ja	7	(87,50%)
nein	1	(12,50%)
Summe	8	
ohne Antwort	0	

22) Sind durch die Bereitstellung der Metadaten auch die gewisse Projektentscheidungen nachvollziehbarer geworden ?

ja	4	(50,00%)
nein	4	(50,00%)
Summe	8	
ohne Antwort	0	

Tabelle 13.2 – Die Grundauswertung des RBMST-Techniker Fragebogens

13.2.4 Management zentrierte Interviewfragenauswertung

Version 1.2.0

universität wien

Interviewfragen für den RBMST-Projektmanager

Diese Befragung dient zur Erhebung der Verwendbarkeit von RBMST, und wird völlig ANONYM erhoben.
Die Interviews dienen dazu um subjektive Intentionen und Interpretationen qualitativ besser erfassen und dementsprechend auch effizienter auswerten zu können.

Die Fragestellungen sind bewusst prägnant formuliert, und werden während des Interviews im Bedarfsfall erläutert.
Falls zu einer Frage keine Antwort gegeben werden KANN, wird der Vermerk K.A. gesetzt und nicht weiter bewertet.

1. Hat sich durch die Workflowabbildung auch ein nachvollziehbareres Projektbild ergeben ?
   ja bedingt, war für dieses Projekt zwar nicht notwendig (1 Mann), aber die Methodik war wichtig
2. Welche projektspezifischen Abfrageeinstellungen sind am sinnvollsten und warum ?
   maximal die Wahl der Filter, da in diesem Projekt sonst alle Kriterien zusammenfallen (keinen Sinn ergeben)
3. Sind interimistische Abfragen zwecks genauerer Projektstandsfeststellung sinnvoll ?
   hier nicht, ansonsten sicherlich.
4. Wird das Konfliktmanagement durch den Einsatz des Systems beeinflusst ?
   nein
5. Erzeugt das System eine negatives Gefühl hinsichtlich der Projekttransparenz ?
   im Gegenteil, es gibt eine schnelle Übersicht der gefragten Informationen
6. Sind durch die Nutzung des Systems verstärkt gruppeninterne Konflikte entstanden ?
   nein
7. Haben sich durch die Systembenutzung auch Kommunikationskanäle zwecks Informationsbeschaffung verändert ?
   nein

Ing. Mag. Marcello Presulli 2014

Version 1.2.0

Interviewfragen für den RBMST-Projektmanager Seite 2

8. Könnte das System andere, auch persönliche, Informationsschnittstellen verdrängen ?

kann ich nicht sagen, glaube aber schon

9. Entstehen durch die Benutzung des Systems auch psychosoziale Effekte (Stress,Erleichterung), wenn ja welche ?

k.A.

10. Bringt das System auf überregional verteilte Entwicklungen einen Mehrgewinn ?

sicherlich, durch die vereinheitliche Darstellung der Systemzeit

11. Wäre die Definition einer eigenständigen Begriffswelt (Ontologie) von Vorteil ?

k.A.

Die hier notierten Zusammenfassungen vermitteln meine getätigten Intentionen völlig korrekt.

firmenmäßige Zeichnung

TEL. 02252/60 71 00-0
FAX 02252/60 71 00-3010
KLINGER
Gebetsroither
GmbH & Co KG
Am Kanal 8-10
A-2352 Gumpoldskirchen

Ich bedanke mich vorallem in meinen Namen, und im Namen der Universität Wien für die tatkräftige Unterstützung seitens der Privatwirtschaft.

Ing. Mag. Marcello Presulli 2014

Abbildung 13.1 – Beispielauswertung eines Interviews für den RBMST-Manager

13.2.5 Technisch orientierte Interviewfragenauswertung

Version 1.2.0

universität wien

Interviewfragen für den RBMST-Projekttechniker

Diese Befragung dient zur Erhebung der Verwendbarkeit von RBMST, und wird völlig ANONYM erhoben.
Die Interviews dienen dazu um subjektive Intentionen und Interpretationen qualitativ besser erfassen und dementsprechend auch effizienter auswerten zu können.

Die Fragestellungen sind bewusst prägnant formuliert, und werden während des Interviews im Bedarfsfall erläutert.
Falls zu einer Frage keine Antwort gegeben werden KANN, wird der Vermerk K.A. gesetzt und nicht weiter bewertet.

1. Hat sich eine bedingte Workflowabbildung negativ ausgewirkt, wenn ja warum ?

   nein, war für mich persönlich gesehen, schon ein längerer Wunsch.

2. Wurden korrekte Schlüsselwortpramatiken bewusst nachträglich entfernt ?

   nein

3. Gibt es interne Konsequenzen bei Nichtbefolgung der Konventionen ?

   nein, es gibt dzt. keine Kontrolle, außer die eigene

4. Welcher funktionale Bereich hat die Konventionsanalyse vollzogen, und wie ?

   normalerweise einer von uns (Techniker), aber haben es nicht im offiziellen Rahmen gemacht

5. Wurde das Wissensmanagement durch den Einsatz des Systems beeinflusst ?

   ja, wir werden jetzt eines evaluieren

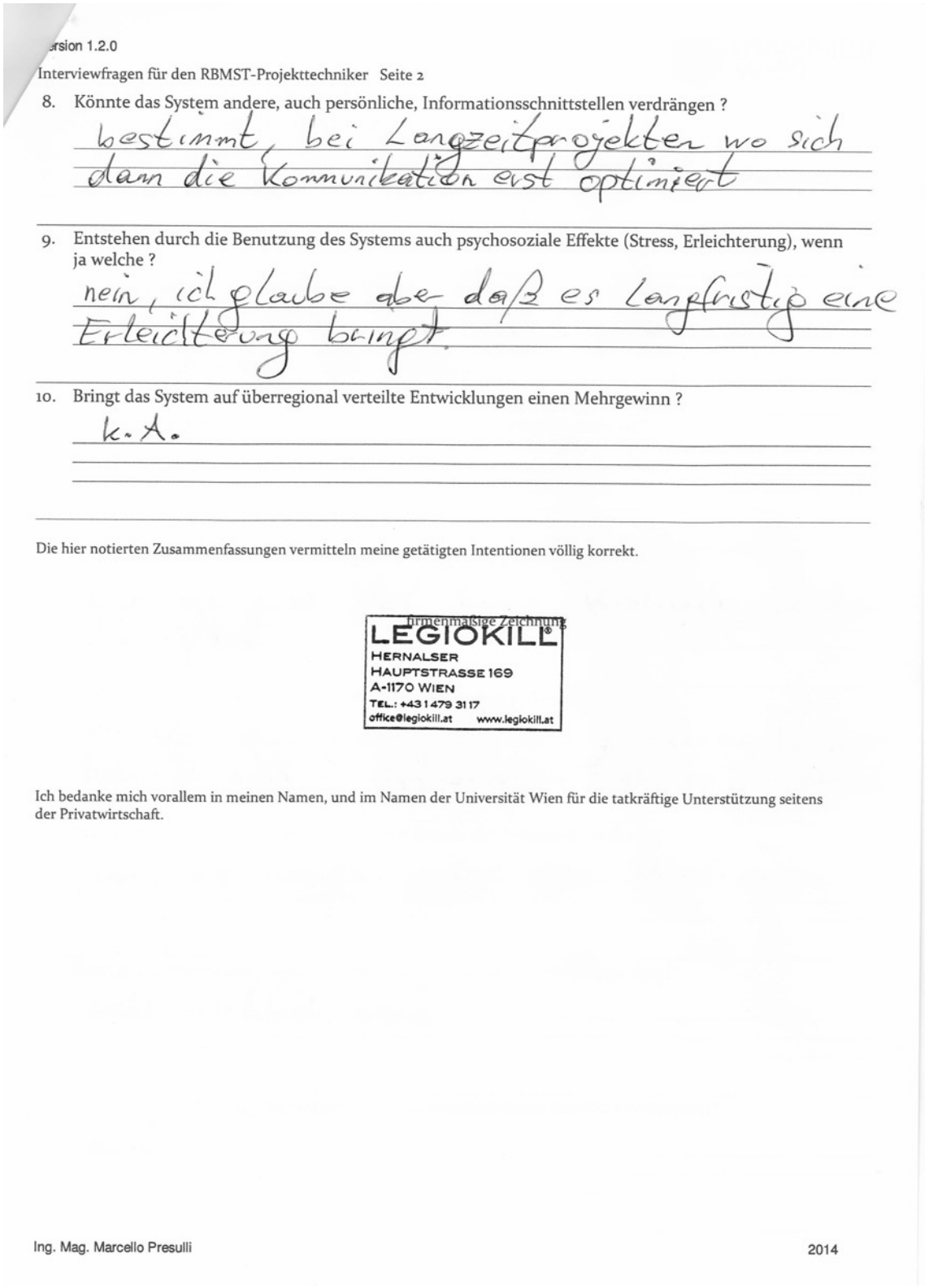
6. Erzeugt das System eine negatives Gefühl hinsichtlich der Projekttransparenz ?

   nicht wirklich, nein

7. Sind durch die Nutzung des Systems verstärkt gruppeninterne Konflikte entstanden ?

   nein

Ing. Mag. Marcello Presulli 2014



ersion 1.2.0

Interviewfragen für den RBMST-Projekttechniker Seite 2

8. Könnte das System andere, auch persönliche, Informationsschnittstellen verdrängen ?

bestimmt, bei Langzeitprojekten wo sich dann die Kommunikation erst optimiert

9. Entstehen durch die Benutzung des Systems auch psychosoziale Effekte (Stress, Erleichterung), wenn ja welche ?

nein, ich glaube aber daß es langfristig eine Erleichterung bringt.

10. Bringt das System auf überregional verteilte Entwicklungen einen Mehrgewinn ?

k.A.

Die hier notierten Zusammenfassungen vermitteln meine getätigten Intentionen völlig korrekt.

firmenmäßige Zeichnung

LEGIOKILL®
HERNALSER
HAUPTSTRASSE 169
A-1170 WIEN
TEL.: +43 1 479 31 17
office@legiokill.at www.legiokill.at

Ich bedanke mich vorallem in meinen Namen, und im Namen der Universität Wien für die tatkräftige Unterstützung seitens der Privatwirtschaft.

Ing. Mag. Marcello Presulli 2014

Abbildung 13.2 – Beispielauswertung eines Interviews für den RBMST-Techniker

13.2.6 Grundauswertung sämtlicher Fragebögen

Manager-Frage	*ja*	*nein*	*Anzahl*	*Art oder Tendenz*
1	–	–	4	Meetings
2	–	–	3	Ausgabeartefakte
3	–	–	3	SVG-Graphen
4	5	3	–	ja
5	–	–	2 von 8	Ausfälle
6	4	4	–	indifferent
7	4	4	–	indifferent
8	5	3	–	ja
9	8	0	–	ja
10	6	2	–	ja
11	3	5	–	nein
12	8	0	–	ja
13	5	3	–	ja
14	8	0	–	ja
15	6	2	–	ja
16	7	1	–	ja
17	1	7	–	nein
18	4	4	–	indifferent
19	6	2	–	ja
20	6	2	–	ja
21	5	3	–	ja
22	5	3	–	ja
23	5	3	–	ja

Tabelle 13.3 – Bewertung der Fragebögen des RBMST-Managers

Techniker-Frage	*ja*	*nein*	*Anzahl*	*Art oder Tendenz*
1	–	–	1	Konsistenz-analysen
2	–	–	4 von 9	Fehleingaben
3	–	–	4	Schlüsselwort state
4	–	–	6 von 8	Dauer <= 90 Sekunden
5	4	4	–	indifferent
6	6	2	–	ja
7	5	3	–	ja
8	6	2	–	ja
9	5	3	–	ja
10	3	5	–	nein
11	4	4	–	indifferent
12	8	0	–	ja
13	8	0	–	ja
14	0	8	–	nein
15	6	2	–	ja
16	5	3	–	ja
17	5	3	–	ja
18	7	1	–	ja
19	6	2	–	ja
20	4	4	–	indifferent
21	7	1	–	ja
22	4	4	–	indifferent

Tabelle 13.4 – Bewertung der Fragebögen des RBMST-Technikers

13.2.7 Grundauswertung sämtlicher Interviewfragen

Manager-Frage	*ja*	*nein*	*indifferent*	*Tendenz*
1	5	3	0	ja
2	4	2	2	ja, Datum & Task
3	7	1	0	ja
4	2	6	0	nein
5	0	6	2	nein
6	3	5	0	nein
7	3	5	0	nein
8	4	0	4	indifferent
9	4	2	3	ja, positiv
10	5	0	3	ja
11	4	2	2	ja

Tabelle 13.5 – Bewertung der Interviewfragen des RBMST-Managers

Techniker-Frage	*ja*	*nein*	*indifferent*	*Tendenz*
1	1	5	2	nein
2	3	5	0	nein
3	0	8	0	nein
4	3	4	1	nein
5	4	3	1	ja
6	2	6	0	nein
7	2	6	0	nein
8	4	1	3	ja
9	3	2	3	indifferent
10	4	0	4	indifferent

Tabelle 13.6 – Bewertung der Interviewfragen des RBMST-Technikers

13.3 Die Hypothesenbewertung

Folgende Notation wird in den nachfolgenden Bewertungen verwendet um die Antworten der diversen Fragestellungen kompakter darzustellen und vereinheitlichen zu können:

— Buchstabe **M** für Manager, **T** für Techniker

— Buchstabe **F** für Fragebogen, **I** für Interview

— Die Zahl ist die fortlaufende Item-Nummer im jeweiligen Bogen

Die anschließende Hybridisierung erfolgt, abhängig von der Art der Frage, entweder durch eine Feststellung der prozentuellen Anteile bezogen auf die Grundgesamtheit, des arithmetischen Mittelwertes und Medians der Fragebogenantworten, sowie einer zusätzlichen Bewertung der Interviewantworten, wobei bei eindeutigen Ja-Nein Antworten ein direkter Bezug hergestellt wird, andernfalls erfolgt eine individuelle Gewichtung durch Interpretation und Erfahrung aus der teilnehmenden Beobachtung. Die Hypothesen 1, 4, 5 und 8 gelten für beide Parteien, den Manager und Techniker, zugleich zu testen, da sich bei den beiden Kohorten Veränderungen einstellen können. Die restlichen Hypothesen 2, 3, 6 und 7 sind vorwiegend Management zentriert, da es hier um die Nutzenfeststellung des Projektmanagements geht und weniger um die technische Verwendbarkeit der Forschungssoftware. Die Ergebnisse der Auswertungen verwerfen, bestätigen oder nehmen lediglich einen indifferenten Status zur Hypothese ein.

Zuallererst lässt sich an den angeführten Rohdaten und den Tabellen 13.3 & 13.4 ableiten, dass die Fragebögen erfreulicherweise vollständig beantwortet wurden, wobei eine projektleitende Person eines Industriepartners während der Erprobungsphase für ca. 10 Tage erkrankt war und somit nicht in der gänzlichen Erhebungszeit zur Verfügung stand. Bei den Interviews, wie die konsolidierten Bewertungen in Tabelle 13.5 & 13.6 vermitteln, wurde der Großteil der Fragen anstandslos beantwortet, wobei bei dieser Befragungsform auch bewusst unspezifische und teilweise gar keine Stellungnahmen seitens der Probanden gegeben wurden.

Des Weiteren wurden auch fallweise Freitext Angaben zu den jeweiligen Fragestellungen angeführt, die die Aussagen teilweise bereichern. So wurde beispielsweise wie unter MF21 ersichtlich, der Kommentar *„bessere farbliche Zuordnung“* abgegeben. Auffällig hingegen sind einerseits die Antworten zur Frage TF1, der konventionellen Konsistenzanalyse, die offensichtlich nur in einem Unternehmen tatsächlich veranstaltet wurde. Andererseits wurde unerwartet oft wie TF2 vermittelt, eine *Selbstkorrektur der Schlüsselpragmatiken* angewandt.

13.3.1 Hypothese – Minderung semantischer Inkompatibilitäten

Zur Überprüfung dieser Aussage werden MF8+MF18+MF22+TF8+TF22 herangezogen. Gemäß den angeführten Grundauswertungen, zeichnet sich aus der Bewertungssicht der Techniker eine Besserung der semantischen Inkompatibilität ab. Die Hypothesenannahme wird somit *bestätigt*.

13.3.2 Hypothese – Erhöhte Informationsverfügbarkeit

Zur Überprüfung dieser Aussage werden MF4+MF5+MI7 herangezogen. Gemäß den angeführten Grundauswertungen, zeichnet sich aus der Bewertungssicht der Manager eine erhöhte Informationsverfügbarkeit ab. Die Hypothesenannahme wird somit *bestätigt*.

13.3.3 Hypothese – Schnellere Informationsexploration

Zur Überprüfung dieser Aussage werden MF13+MF14+MI3 herangezogen. Gemäß den angeführten Grundauswertungen, zeichnet sich aus der Bewertungssicht der Manager eine schnellere Informationsexploration ab. Die Hypothesenannahme wird somit *bestätigt*.

13.3.4 Hypothese – Verbesserte Informationsrepräsentation

Zur Überprüfung dieser Aussage werden MF9+MF21+TF12+TI1 herangezogen. Gemäß den angeführten Grundauswertungen, zeichnet sich aus der Bewertungssicht der Manager und Techniker eine Verbesserung der Informationsrepräsentation ab. Die Hypothesenannahme wird somit *bestätigt*.

13.3.5 Hypothese – Höhere Informationstransparenz

Zur Überprüfung dieser Aussage werden MF10+MF12+MI5+TF10+TI6 herangezogen. Gemäß den angeführten Grundauswertungen, zeichnet sich aus der Bewertungssicht der Manager und Techniker eine höhere Informationstransparenz ab. Die Hypothesenannahme wird somit *bestätigt*.

13.3.6 Hypothese – Effizientere Kommunikationspfade

Zur Überprüfung dieser Aussage werden MF17+MF20+MI8+TI8 herangezogen. Gemäß den angeführten Grundauswertungen, zeichnet sich aus der Bewertungssicht der Manager ein verneinen, bei den Technikern ein bejahen ab.

Sie bejaht, wie auch die Beantwortung von MF20 suggeriert, vielmehr ein gewünschtes Frühwarnsystem, als ein Wegrationalisieren von Informationsstellen. Hier wurde offensichtlich die Frage MF17, bedingt durch eine offensichtliche Mehrdeutigkeit, missinterpretiert – Die Frage hätte vielmehr als *ein effizienter Ersatzpfad bei temporärem Informationsträgerausfall* formuliert werden sollen. Die Hypothesenannahme ist als theoretisch *indifferent.* Wie in 14.3.3 jedoch angeführt, ist die Hypothese aufgrund einer überwiegenden Missinterpretation praktisch als *verworfen,* anzunehmen.

13.3.7 Hypothese – Effektivere Informationssicherheit

Zur Überprüfung dieser Aussage werden MF6+MF7+TF1+TF2+TI2 herangezogen. Gemäß den angeführten Grundauswertungen, zeichnet sich aus der Bewertungs-

sicht keine signifikante Erhöhung der Informationssicherheit ab. Es wird darauf hingewiesen, dass die vollautomatische Bewertung des Managerfragebogens hier FEHLEHAFT ist.

Das Ergebnis entstand nach einer irrtümlichen Bedienung. Das reelle Gesamtergebnis der Antworten wäre 50% Nie und 50% <=2-mal. Des Weiteren wurde auch hier der ursprüngliche Begriff der Informationssicherheit gedehnt, da die Sicherheit in gewisser Maßen auch auf der Echtheit der Information beruht, die hier aber nicht gewährleistet werden kann, da prinzipiell keine objektive Konsistenzanalyse angestoßen wurde um den Informationsgehalt zu verifizieren. Die Hypothesenannahme wird in dieser Hinsicht somit *verworfen*.

13.3.8 Hypothese – Psychosoziale Entlastungseffekte

Zur Überprüfung dieser Aussage werden MF15+MF16+TF14+TF20+MI9+TI9 herangezogen. Gemäß den angeführten Grundauswertungen, zeichnet sich aus der derzeitigen Bewertungssicht ein langfristig zu erwartender leicht positiver Entlastungseffekt ab. Die Hypothesenannahme wird somit *bestätigt*.

13.3.9 Allgemeine Fragestellungen

Wie bereits in Kapitel 11.1 erläutert wurde, handelt es sich bei dieser Form der Ermittlung um eine Querschnittsstudie die verstärkt gewisse Teilaspekte wie sie in den oben diskutierten Hypothesen formuliert wurden expliziert studiert.

Der zweite Teil dieser Studie behandelt andere sekundäre Fragestellungen um auch ein holistisches Gesamtbild nach der ersten Iteration der Erprobungsphase zu erhalten. Basierend auf den Ergebnissen kann ein weiterführendes vertiefendes Ermittlungsverfahren angestrebt werden, beziehungsweise lassen sich anhand der Ergebnisse bereits technische Notwendigkeiten ableiten.

13.3.9.a Ist ein separates Hilfstool zur Metadateninjektion sinnvoll ?

Diese Frage kann durch TF4+TF13+TF15+TF16+TF17+TF18+TI4 beantwortet werden. Die Antworten ergeben mehrheitlich eine klare Tendenz zur *Bejahung* der Frage.

13.3.9.b Wäre die derzeitige technische Realisation bereit für einen Produktiveinsatz ?

Diese Frage kann durch MI2+TF5+TF6+TF7+TF9 beantwortet werden. Die Antworten ergeben mehrheitlich eine Tendenz zur *Bejahung* der Frage.

13.3.9.c Deckt die derzeitige These mit ihren Formulierungen alle gedachten Einsatzeventualitäten genügend ab ?

Diese Frage kann durch MF11+MI11+MI2+TF3+TF11+TF21 beantwortet werden. Die Antworten ergeben mehrheitlich eine Tendenz zur Ver*neinung* der Frage.

13.3.9.d Beeinflusst der Einsatz von RBMST Informationssysteme, speziell Wissenssysteme ?

Diese Frage kann durch MF19+MI2+TI5 beantwortet werden. Die Antworten ergeben mehrheitlich eine Tendenz zur *Bejahung* der Frage.

13.3.9.e Könnte ein überregionaler Systemeinsatz hinsichtlich des Projektmanagements mehr Sinn machen ?

Diese Frage kann durch MI10+TI10 beantwortet werden. Die Antworten ergeben mehrheitlich eine Tendenz zur *Bejahung* der Frage.

13.3.9.f Haben sich unvorhergesehene Verhaltensweisen etabliert beziehungsweise wurden weitere Prozesse angestoßen ?

Diese Frage kann durch MI4+MI6+TF19+TI3+TI7 beantwortet werden. Es haben sich offensichtlich keine Pragmatiken bezüglich der Metadateninjektion etabliert. Es gibt keine klaren Notwendigkeiten zur speziellen Behandlung bestehender Prozesse mit Ausnahme einer etwaigen Erweiterung des Konfliktmanagementprozesses. Die Antworten ergeben eine knappe Tendenz zur Ver*neinung* der Frage.

13.4 Schlussfolgerungen

13.4.1 Generelle Bemerkungen

Die Fragen MF1, MF2 und MF3 dienen der prinzipiellen Einsatzbereitschaft aus der Nutzersicht. Sie decken sich in ihrer Beantwortung dem normalen Nutzungsverhalten ähnlicher Werkzeuge des Projektmanagements. Es kann daraus abgeleitet werden, dass das Einsatzfeld, ohne effektiv strategischen Planungscharakter, in der Spontannutzung bei verschiedenen Projektmeetings liegt.

Hypothese	*Annahme*
1. Minderung semantischer Inkompatibilitäten	bestätigt
2. Erhöhte Informationsverfügbarkeit	bestätigt
3. Schnellere Informationsexploration	bestätigt
4. Verbesserte Informationsrepräsentation	bestätigt
5. Höhere Informationstransparenz	bestätigt
6. Effizientere Kommunikationspfade	verworfen
7. Effektivere Informationssicherheit	verworfen
8. Psychosoziale Entlastungseffekte	bestätigt

Tabelle 13.7 – Ergebnis der konsolidierten Hypothesenbewertungen

Wie die Hypothesentests in Tabelle 13.7 zeigten, sind 6 von 8 Aussagen (75%) positiv bestätigt worden was den Anlass zur Weiterverfolgung des Systems grundsätzlich bestärkt, die restlichen zwei Aussagen sollten verstärkt betrachtet werden. Bei ihnen drängt sich die Frage auf, ob ein konsequentes Weiterverfolgen überhaupt Sinn macht oder die korrespondierenden Fragestellungen eindeutig formuliert wurden, beziehungsweise überhaupt die Nutzbarkeit dieses Teilaspekts gegeben ist.

Die Interpretationen der Interviewantworten, und die nachfolgend unter 13.4.2 informell beschriebenen Erfahrungswerte aus der Feldforschung zeigen auch Extremsituationen auf, wo das System nicht zwingenderweise an die Grenzen der Einsatzfähigkeit, sondern vielmehr an den tatsächlichen *Grenznutzen* stößt. So decken beispielsweise Unternehmen mit nur einer vollzeitbeschäftigten Entwicklungskapazität implizit auch negative Skaleneffekte auf, die den sinnvollen Einsatzbereich von RBMST nach unten hin abgrenzen.

Nach oben hin zeichnen sich keine offensichtlichen Limitationen ab, trotz allem ergeben sich auch hier theoretische Einsatzgrenzen, bedingt durch eine zu stark technisch inkompatible Projektlandschaft, die jedoch mittels softwaretechnischer Anpassungen und Erweiterungen von RBMST beherrscht werden könnten.

13.4.2 Erkenntnisse der partizipativen Feldforschung

13.4.2.a Sinnvolle Workflow Abbildungen

Bei vielen Probanden Unternehmen mussten mehr oder weniger tiefgreifende strukturelle Maßnahmen im Bereich der Abbildung eines adäquaten Workflows in Bezug auf die vorherrschende Repositorystruktur ergriffen werden. Diese Tätigkeit war im Vorfeld der tatsächlichen Erprobungsphase zu leisten. Zur Sicherheit wurde der Letztstand der bestehenden Repositorystruktur gesichert um bei Bedarf später wieder nahtlos aufsetzen zu können.

So wurde bei der Hälfte der Partner ein sinnvolles Branching, sprich eine mäßige Aufteilung der Projekthauptaufgaben auf wohlbenannte Zweige eingeführt, beziehungsweise auch ein einheitliches Namensschema für sämtliche Autoren und ihre aktiven Kontaktadressen erstellt. Natürlich wurden nach diesen basalen Änderungen auch mögliche weitere Projektunterteilungen über den <Task> Tag in der zentralen Konfigurationsdatei empfohlen. Insgesamt wurde während der Umstrukturierung bereits bemerkt, dass sofern die Notwendigkeit auch verstanden wurde, sich ein positives Engagement einstellt. Die Nutzer wie auch Benutzer haben allmählich die Sinnhaftigkeit einer logischen Untergliederung der Hauptprojekttätigkeiten verstanden, wobei die Art der semantischen Gliederung den Unternehmen gänzlich überlassen war. Der Effekt der Umstrukturierung widerspiegelte sich auch in der Antwort der Interview-Fragen, vor allem der Techniker selbst, wo die sogar negativ formulierte erste Fragestellung mit überwiegend positiven Eindrücken beantwortet wurde. Jedoch ob und wie tiefgreifend die Projektaufgliederung betrieben wird, hängt gänzlich von der Unternehmenskultur und den internen Unternehmensstrukturen selbst ab.

13.4.2.b Insgesamt verbesserte Teamfähigkeit

Die interessanteste gruppenzentrierte Beobachtung konnte im Bereich der Kollaboration festgestellt werden. Hier kamen speziell meine erworbenen Erkenntnisse aus dem iCom-Projekt zugute. Es fand ein vorerst nur sehr zaghaftes, und dann immer kollektiveres, quasi synchrones, Miteinander in der gemeinsamen Injektion von Metadaten statt.

Dieses Verhalten, das ich nicht erwartete, kam deswegen zustande, da die Techniker die Injektionen nach logischen Kategorien durchführten, ein Vorteil wie auch Nachteil des asynchronen Lesezeichenmechanismus. Ein Vorteil, da die Eingaben bis zum nächsten commit zeitlich frei disponierbar sind. Ein Nachteil deswegen, weil der Autor der Injektion nicht mit dem Autor des darunterliegenden Changesets korrespondieren muss, ergo der Wahrheitsgehalt in Frage gestellt werden kann. So waren zwar alle gleichzeitig mit der Injektion der *state* Schlüsselwortpragmatik beschäftigt, denn jeder wollte offensichtlich seine Arbeitszeit am Projekt selbst dokumentieren, aber die Injektionen für enquiry & knowledgeSource fiel in die Domäne einer dedizierten Person.

Bei den *dependency* Schlüsselwortpragmatiken hingegen gab es manchmal einen Verantwortlichen manchmal tat es jeder gleichverteilt. Bezüglich Hilfsstellung war jeder auffällig bereit und auch bemüht, den anderen zu unterstützen. Hier wurde offensichtlich versucht den Stand im Bereich des Metawissens und der Medienkompetenz [DN2000] zu vereinheitlichen, sowie eine initiale Wissensbasis für nachfolgende Operationen zu entwickeln. Es entwickelten sich durch die Metadaten Injektion bedungene Informationsaustauschszenarien, die es in dieser Tiefe zuvor sicherlich nicht gegeben hat, da man sich plötzlich gemeinsam Gedanken dazu machen musste, wie man eigentlich beispielsweise ein Wiki als Wissensaustauschplattform am besten strukturiert – ein Thema, das sich nicht jedem vorher stellte.

Diese und ähnliche Effekte konnte auch durchgängig in den iCom-Workshops[1] beobachtet werden, wo kleinere Gruppen sich plötzlich, mit ihnen teilweise fremden Aufgabestellungen, kollaborativ auseinandersetzen mussten um eine gemeinsame agile Vorgehensweise zu entwickeln [IW2013]. Folgendes Zitat festigt den Zusammenhang des vitalen Informationsaustausches und der gesteigerten Teamfähigkeit:

„Die geographische Verteilung, die mediengestützte, oft zeitverzögerte Kommunikation und die Herkunft von Teammitgliedern unterschiedlichster Kulturen und Organisationen haben gravierende Einflüsse auf den Informationsaustausch und das gemeinsame Wissen von Mitgliedern virtueller Teams. Gemeinsames Wissen bezieht sich in diesem Fall auf das Wissen, das beteiligte Teammitglieder teilen und von dem sie auch wissen, dass sie es teilen.

Diese gemeinsame Wissensbasis stellt eine Grundlage für erfolgreiche Kommunikation und Koordination dar, da ohne gemeinsame Wissensbasis die Gesprächsgrundlage für Kommunikationspartner unterschiedlich ist. Dadurch können Missverständnisse und Fehlinterpretationen entstehen, die zu Behinderungen im

[1] http://www.icomproject.eu/archive/phd-seminars/practice-research-workshop

Arbeitsablauf und zu Konflikten führen können. Für virtuelle Teams existieren deshalb drei relevante von Informationen, die unter Teammitgliedern während des Projektablaufes ausgetauscht werden müssen, um eine gemeinsame Wissensbasis aufzubauen und aufrecht zu erhalten: Aufgabenbezogene, soziale und Kontextinformationen.“[1]

13.4.2.c Kreative Referenzierungsmethoden

Manche Personen in den Entwicklungsteams haben sich erst im Zuge mit der Einführung der neuen Technologie, auch eindringlicher mit dem darunterliegenden VCS beschäftigt. Speziell bei den knowledgeSource Schlüsselwortpragmatiken wurde offensichtlich mangels konventioneller Einigung, oder geringem Selbstengagement *Selbstreferenzierungstechniken* eingeführt. Diese Referenzen waren eine reine interne Changeset Nummerierung, und stellten somit keine externen Bezüge dar. Sie stellten zumeist fortlaufende Revisionsnummern des zugrundeliegenden Repositoriums dar. So wurden zumeist aus Bequemlichkeit Reifikationen gebildet, die sich auf eine andere Revisionsnummer beriefen, um einen zuvor schon im Quellcode vermerkten Sachverhalt oder Wissensquelle nicht nochmals in ein Wissensmanagementsystem überführen zu müssen. Eine Schlüsselwortpragmatik aus der Praxis soll beispielhaft die verwendete Syntax verdeutlichen.

SMAC.knowledgeSource(336)

Diese Angabe bezog sich ohne weiteren Vermerk auf eine Revisionsnummer des darunter liegenden lokalen Repositories. Wenn unter dieser Revisionsnummer der Log-Eintrag des zugrundeliegenden *commits* begutachtet wird, stellte man fest, dass diese Angabe eine Referenz auf die „Ragel Dokumentation Version 6.8 Sektion 5.8.3“ darstellt. Nicht nur, dass solche Angaben mühsam in der Rückverfolgung sind, sind sie auch unzuverlässig und gegebenenfalls volatil, da sich die interne Revisionsnummer im Gegensatz zum Changeset-Hashwert beim Verschmelzen (merge) verschiedener Zweige auch ändern kann.

Ein ähnliches Beispiel war auch im Fall einer *enquiry* Schlüsselwortpragmatik zu erkennen, wo zwar die Rückmeldung einer Anfrage konventionell via Email vereinbart wurde, aber keine Vorgaben bezüglich eines gemeinsamen Anfragekatalogs oder vergleichbarem gemacht wurden. Hier wurde wiederum ein Zeitstempel in Form von „*Datum_Uhrzeit*“ als *enquiry* Argument mangels anderer vereinbarter Alternativen missbraucht, was natürlich wiederum nur begrenzt Sinn machte, da die Information rein lokal gebunden war.

Wie an diesem und vorigen Beispiel gut zu erkennen ist, können sich, mangels unzureichender Konventionsvereinbarungen oder auch ungenügender Motivation, (Hilfs)techniken etablieren die einerseits so nicht erdacht und auch keinesfalls erwünscht sind.

---

[1] Stöber 2008: Kommunikations- und Koordinationsbedarfe in verteilten DV-Projekten

13.4.3 Selbstkritische Betrachtung der Ermittlungsverfahren

Der Ansatz der diversen Frageformulierungen hatte die Ausrichtung auf eine Querschnittstudie mit vertieften Teilaspekten. Dies erschien als ein probater Mittelweg hinsichtlich der zur Verfügung stehenden Zeitspanne der Unternehmen und dem Wunsch ganzheitlicher Erfassung interessanter Themen um in einer theoretisch nachfolgenden Erprobungsphase jene herausgearbeiteten Aspekte spezieller zu behandeln.

Das Problem das sich bei dieser Art der Ermittlung stellt ist, dass die Summe der dazu benötigten Fragen enorm wäre, was wiederum in diesem Rahmen nicht möglich war. Es zeigte sich auch, dass teilweise Interviewfragen aufgrund zu kurzer Erprobungsphase nicht eindeutig bis gar nicht beantwortet wurden. Eine Langzeitstudie hätte in diesem Zusammenhang sicherlich jene Fragen eindeutiger erörtern können.

Es wurden auch Bemühungen unternommen die Fragen einerseits neutral und prägnant zu gestalten, andererseits aber auch nicht zu simpel um einfache Entscheidungsfragen weitgehend zu vermeiden. Das Ziel war letztendlich, durch Likertskalen und Maßzahlen, richtungsweisende Tendenzen für ein weiteres Vorgehen zu ermitteln. Die Schwierigkeit darin ist die treffende Formulierung zu finden, welche aus subjektiver Sicht oft gelang aber definitiv nicht immer, da vor allem bei Interviewfragen oft Rückfragen gestellt wurden, da die Probanden die Frage anscheinend anders aufgefasst hatten als ursprünglich erwartet. Hier hätte vor allem eine Vorstudie mit 1 - 2 Testpersonen, erwartungsgemäß eine effektive Verbesserung hinsichtlich eindeutiger Frageformulierungen ergeben.

Auch auf den Einsatz der Fragebogensoftware ist Bezug zu nehmen. Insgesamt ist das Werkzeug professioneller Machart jedoch gibt es hinsichtlich der Bedienung Eigenheiten. So ist ein Zurückblättern bereits beantworteter Fragen offensichtlich problematisch, da sich manchmal bereits getätigte Wertzuweisungen ändern, die dann wiederum neu gesetzt werden müssen. Auch die Dokumentation ist nicht lückenlos und lässt manche Frage offen, wo es dann gilt, die Antwort im experimentellen Eigenstudium selbst herauszufinden. Trotz allem ist die elektronische Erfassung eine äußerst effektive und exakte Methode, um Fragestellungen wissenschaftlich nachvollziehbar zu behandeln.

Teil V
Zusammenfassung

14 Kapitel

Diskussionen und weitere Erörterungen

„Der beste Weg, die Zukunft vorauszusagen,
ist, sie zu gestalten.“
Willy Brandt

Schwerpunkte in diesem Kapitel:

- Diskussion
- Behandlung offener Themen
- Grenzen der Studie
- Ausblick
- Schlusswort

14.1 Diskussion

Die Resultate der konkreten Untersuchungen wurden bereits in Kapitel 13.5 ausführlich im Kontext der Hypothesenverifikationen diskutiert und zu guter Letzt wurde versucht verschiedene Konklusionen daraus zu ziehen. Die Effekte und Folgen die sich dadurch abzeichneten waren für sich betrachtet bereits sehr interessant, doch der Bogen den das behandelte Dissertationsthema insgesamt umspannt, ist weit größer. Die Diskussion begann bei verwandten Arbeiten und entwickelte sich weiter in einem knappen geschichtlichen und kategorischen Überblick der in der Software-Entwicklung verwendeten Supportsysteme. Es wurden in diesem Zusammenhang Motivationen dargelegt, die mehrfachen Ursprungs sind, und auch implizite Effekte und Gesetzmäßigkeiten die sich im Zuge eines Projektverlaufes auch nachhaltig manifestieren können. Über diese Thematisierung wurde ein These gesponnen, die durch eine Implementation auch in der Praxis erprobt werden soll, jedoch nicht im Sinne einer vollständigen Produktivitätsstudie, sondern den Stellenwert eines *„proof of concepts“* eingenommen hat.

Um auch den *common sense* der Anwenderschaft gänzlich einbeziehen zu können, wurden die Anforderungen an die zu erstellende Software nicht selbst erdacht, sondern vielmehr kollaborativ erstellt. Zuvor war es auch notwendig den Beteiligten die Intentionen der konkreten Bestrebung plakativ zu vermitteln, nämlich dass es prinzipiell um *ihren* Unternehmens- beziehungsweise Projektnut-

zen geht, und eigentlich das Artefakt ein Werkzeug zur erleichterten Bewältigung der Projektkomplexität darstellt, und das für beide Seiten der Projektdomäne.

Die These wurde somit in ihren Teilen plastisch vorgeführt und durch den Diskurs auch wiederum angereichert, teilweise wurden dadurch sogar Lösungsansätze verworfen, man denke nur dabei an das Ansinnen einer DSL, aber es kristallisierten sich auch konsistente Weg zur Bewältigung der verschiedensten Anforderungen heraus. Ein gutes Beispiel wäre hier die Verwendung des bestehenden RDF->SVG Transformationspfades samt der dazu notwendigen Toolchain. Die kollaborativen Sitzungen gaben die grundsätzliche Richtung vor, die es bei der Erstellung der Software zu berücksichtigen gab.

Somit sollte eine weiterführende Diskussion dort anknüpfen wo die Realisation der Forschungssoftware endet, nämlich dem kollaborativen Erfassen jener benötigten Anforderungen die die Implementation eines Werkzeuges vorgeben, das auch im Praxisfeld getestet werden kann.

Nachträglich gesehen, haben sich im Zuge dieser Arbeit etliche Verbesserungsvorschläge etabliert, die jedoch wie so oft bei einem Vorhaben, niemand der Beteiligten im Vorfeld absehen konnte. So sind nicht nur technische Belange die während der Erprobungsphase identifiziert wurden, sondern auch generell implementationsübergreifende Themen die auch Rückwirkungen auf die These haben, werden exemplarisch im nächsten Kapitel vorgestellt.

Ein weiterer Diskussionspunkt stellt sich sicherlich auch in Bezug auf die Erprobungsphase. Da gibt es einige Aspekte die ausführlicher betrachten hätten werden können und welche die *ex post* gesehen gar nicht so essentiell waren wie vorerst angenommen. Wichtig dabei ist immer die Ausrichtung beziehungsweise der Fokus der Studie. Hier war definitiv, da die These in ihrer Implementationsform in einem Unternehmenskontext so erstmalig zur Anwendung gekommen ist, noch einiges unklar und dementsprechend sicherlich auch nicht so zweckbestimmt. Diese Studie hatte eher den Charakter einer breiten Querschnittsstudie, auch mit dem Hintergrund persönliche Erfahrungen in diesem Zusammenhang zu sammeln. Wie bei einer solchen Studie eben der Fall ist, werden viele Facetten diskutiert ohne jedoch eine Spezialisierung zu erfahren. Dies hat den entscheidenden Vorteil einerseits die Sinnhaftigkeit der aufgestellten These bewerten zu können und auch die zeitliche Dimension der Erprobungsphase besser einplanen zu können. So gibt es prinzipiell kein synchron verpflichtendes Testintervall wie es bei Lang bzw. Kurzzeitstudien der Fall ist, sondern vielmehr einen unternehmensspezifischen Redaktionsschluss, der dann eingeleitet wird, wenn die notwendigen Beobachtungen angestellt wurden.

In diesem Zusammenhang möchte ich die Diskussion mit den konkret identifizierten Themen die sich während der Einsatzphase explizit ergeben haben, sowie einigen Punkten, die sich implizit aus den Ergebnissen der in Kapitel 13.2.6 diskutierten Antworten der Fragebögen und Interviews ableiten, schließen. Kombiniert mit den subjektiven Beobachtungen und persönlichen Gespräche mit den Einzelnen wie auch in der Gruppe, ergaben sie erst ein Gesamtbild gemäß der Tradition der Feldforschung.

14.2 Behandlung offener Themen

Der behandelte Themenkomplex ist sehr weitreichend, da wie sich an mehreren Stellen in der Diskussion dieser Arbeit bereits gezeigt hat, Interessen aufeinander prallen die teilweise heterogener Natur sind. Es werden dabei Intuitionen formuliert die einerseits aus der Sicht der Ökonomie kommen aber auch teilweise reinen organisationstechnischen Charakter besitzen. Technische Belange erwiesen sich zumeist *ex post* in der Erprobungsphase, können deswegen in Bezug auf Änderungsanforderungen in klarer Form abgegrenzt und formuliert werden.

Im Zuge dieser Erprobung haben sich nun mannigfaltige Wünsche und Optimierungspotentiale erschlossen die hier behandelt werden. Die diversen Anforderungen werden zuerst kategorisiert um sie im weiteren Verlauf besser thematisieren zu können. Einige dieser Themen wurden vom Autor selbst erkannt, wobei der Großteil der Empfehlungen definitiv seitens der Industriepartnerschaft kam. Die folgenden Themen werden gemäß der relativen Notwendigkeit bewertet. So ergeben sich aus dieser Betrachtung zwei Bereiche, nämlich m*andatorisch* und o*ptional,* der die Themen jeweils zugeteilt werden.

14.2.1 Mandatorischer Themenkreis

14.2.1.a Nachträgliche Änderung von Schlüsselwortpragmatiken

Dieses Thema ist sozusagen eines der meist diskutierten, denn die technische Behandlung dieser Forderung entscheidet im Weiteren auch über die Dringlichkeit zusätzlicher Werkzeuge. Dieser Sicherheitsmechanismus kann nämlich prinzipiell auf zweierlei Arten realisiert werden. Die *asynchrone* Lösung ist prinzipiell die einfachere jedoch auch nicht so effektiv wie die *synchrone*, die jedoch ein spezielles Eingabefrontend voraussetzt. Um diese Aspekte weiter zu diskutieren, stellt Tabelle 14.1 die beiden Ansätze kurz gegenüber.

Eigenschaft	*synchron*	*asynchron*
dediziertes Eingabefrontend	ja	nein
gleichbleibende Bedienung möglich	ja	nein
erhöhte Sicherheit gegen Verfälschungen	ja	nein
Mehrwert durch Zusatznutzen	ja	nein
höhere Laufzeiteffizienz	ja	nein
höherer Entwicklungsaufwand	ja	nein

Tabelle 14.1 – Gegenüberstellung der synchronen & asynchronen Lösung

Die Forderung nach Sicherheit leitet sich zwangsläufig aus folgender Überlegung ab. Wenn man eine versehentlich falsch formulierte Schlüsselwortpragmatik über den obligatorischen Lesezeichenmechanismus injektiert und nachträglich richtig stellt ist, stellt das einen formal korrekten Ablauf dar. Es wäre das Selbe als wenn man während einer konventionellen Analysephase die fehlerhaften Metadaten nach

Absprache korrigieren würde. Dank der seiteneffektfreien Zeigerarithmetik stellt das kein hartes Problem dar. Was in diesem Zusammenhang dienlich erscheint, ist gerade im anderen Fall der mutwilligen Änderung nachteilig.

Problematisch wird es, wenn man *bewusst* Schlüsselwortpragmatiken nachträglich verfälscht um gewisse Effekte zu erzielen oder unangenehme Projektdetails vertuschen möchte. Durch geschickte Manipulationen würden sich zum Beispiel Projektabhängigkeiten oder auch Projektzeiten und deren Kostenmetrik sich verfälschen lassen. Übrigens der Punkt bezüglich der Projektzeiten ließe sich auch durch ein bewusstes verstellen der Systemzeit am PC realisieren, welches sich jedoch über einen geeignete firmenspezifische Domänenrichtlinie, wie es bei MS-Windows basierten Systemen üblich ist, verhindern lässt.

Anders formuliert, gilt es zu überlegen wie man eine bewusste nachträgliche Lesezeichenänderung zumindest wirkungsvoll erkennt. Dazu bieten sich gleich mehrere Möglichkeiten schlagartig an. Man könnte zum Beispiel den Hookmechanismus vom Wirtsystem nutzen um bei jeder Bookmarkänderung zu prüfen, ob der Eintrag neu oder bereits bestehend abgeändert wurde. Eine ähnliche Idee knüpft an den PlugIn-Mechanismus an.

Über ein adäquates PlugIn hätte man die Möglichkeit ein dediziertes Kommando zu entwickeln und das bestehende Kommando *bookmark* gleichzeitig zu deaktivieren. Jedoch gilt auch hier wieder zu bedenken, dass spezielle Dateirechte beachtet werden müssen, um die Sicherheit des Mechanismus zu gewährleisten.

Auch ein einfacher Loggingmechanismus könnte auch schon ausreichend sein, der ein nachträgliches ändern von Schlüsselwortpragmatiken in sicherer Form protokolliert und somit den Sachverhalt festhält. Zu guter Letzt wäre natürlich auch die Eingabe über ein geführtes Werkzeug noch am zuverlässigsten. Dieser Punkt wird nochmals an anderer Stelle aufgegriffen, denn die Sinnhaftigkeit eines solchen Tools erhärtet sich zusehends.

14.2.1.b Notifikation bei wissensbasierten Schlüsselwortpragmatiken

Die Fragebogenfrage 19 des RBMST-Managers (MF19) hat gezeigt, dass zwar eine qualifizierte Mehrheit den elektronischen Anfrage-Kommunikationsweg vorzieht, jedoch der Konnex über einen Anfragekatalog als ungenügend erscheint. Darauf basierend entwickelte sich der generelle Wunsch nach automatischer Benachrichtigung wissensbasierter Schlüsselwortpragmatiken. Unter diesem Begriff werden vor allem jene Injektionen verstanden, wo das Schlüsselwort *enquiry* oder *knowledgeSource* zum Einsatz kommt. Über diese beiden Schlüsselwortargumente wird im Fall *knowledgeSource,* prinzipielles Wissen manifestiert, oder aber, wie es beim anderen Schlüsselwortargument zutrifft, Quasiwissen erfragt. Auch hier bieten sich theoretisch einfache Lösungsmöglichkeiten über den Umweg der Mechanismen des Wirtsystems an.

Der Hook-Mechanismus würde in diesem Zusammenhang harmonieren, da über seine einfache Art der Aktivierung ein Skript genügen würde, um in simpler Weise ein Drittprogramm, ähnlich dem *mail* unter Linux, aufzurufen welches über

Kommandozeilenparameter alle wesentlichen Daten an einen bestimmten Adressaten sendet. Natürlich würde wie zuvor schon angeschnitten ein dediziertes Eingabetool auch diese Forderung, im Sinne eines Mehrwertes durch Zusatznutzen, mit Leichtigkeit erfüllen.

Die Schwierigkeiten in diesem Fall sind weniger technischer Natur sondern vielmehr die Randbedingungen, wie wer soll der Empfänger sein, wie soll die Nachricht formatiert werden oder welchen konkreten Titel sollte diese Mail erhalten und wie soll das System reagieren wenn diese Nachricht aufgrund von Netzwerkproblemen nicht gesendet werden kann.

14.2.1.c Kompakte syntaktische Form der Schlüsselwortpragmatiken

Eine Idee, die während der Erprobungsphase immer immanenter wurde, war die kompaktere Gestaltung der Schlüsselwortpragmatiken an sich. Durch eine marginale Umstellung der Notation würden sich einerseits Vereinfachungen in der Metadateninjektion ergeben und andererseits in der RBMST-Software einige Komplexitätsreduktionen von selbst ergeben.

Die einzige Umstellung, die dazu von Nöten wäre ist, dass das Schlüsselwort als erstes Argument dient, und die dazu passende Information als zweites Argument fungiert. Damit hätte die Syntax eine kompaktere auch teilweise intuitiverer Darstellungsform und dennoch würde die Semantik unverändert bleiben. Die folgende Abbildung 14.1 veranschaulicht das mögliche Vorhaben an einem Beispiel.

$$Package1.Task1(dependency, Task2)$$

Abbildung 14.1 – kompaktere Syntaxform der Schlüsselwortpragmatik

Die hier vorgeschlagene Syntax bedarf lediglich einer kollaborativen Entscheidung zur Konvention sowie die Umstellung in der Software, wobei sogar ein Koexistieren mit der derzeit bestehenden Syntaxform vorstellbar wäre.

14.2.1.d Stabilitätsverbesserungen von RBMST

Dieser letzte mandatorische Punkt liegt eigentlich mir selbst am Herzen, da ich in den verschiedensten Einsatzverläufen der Erprobungsphase selbst bemerkt habe, dass hinsichtlich der Stabilitätsverbesserung noch Handlungsbedarf gibt, der jedoch im Vorfeld nicht vakant war und auch dementsprechend nicht berücksichtigt wurde.

Vorrangig gilt es kontinuierliche Verbesserungen im Bereich der Ausnahmebehandlung von RBMST, welche am dringlichsten erscheinen, zu erzielen. Um in heterogenen Unternehmensumgebungen auch tatsächlich sinnvoll verwendbar zu sein hat sich gezeigt, dass das Beherrschen wohldefinierter Transformationsprozesse um die Repository-Kompatibilität zu gewährleisten ebenfalls als unabdinglich erscheint.

So können Datenbankkonvertierungen in horizontaler Ebene, also zwischen zwei unterschiedlichen VCS, notwendig sein um die benötigten Leistungsanforderungen RBMST zu Verfügung zu stellen, denn ansonsten müsste nämlich die Applikation

tiefgreifend adaptieren werden müssen[1]. Auch in vertikaler Ebene, im Migrationspfad des VCS selbst, kann es Anforderungen geben, die jedoch eher den Hintergrund der Fehlerbereinigung oder der Nutzung neuer Funktionen haben.

Insgesamt wurde während der Erprobungsphase teilweise eine mangelnde Stabilität des Werkzeuges beobachtet, wie zum Beispiel die Behandlung des Autornamens im Zusammenhang mit seiner Kontaktadresse, oder nicht wohl geformte Zeitstempel können allesamt das System in eine nicht korrekt behandelte Ausnahmesituation (Exception) bringen. Gerade die konzise idiomatische Behandlungsmöglichkeit über die vertraute *case Kontrollstruktur* von Scala ermöglicht in diesem Bereich eine einfache Vorgehensweise zur weiteren Vervollständigung.

14.2.2 Optionaler Themenkreis

14.2.2.a Eingabewerkzeug *CreateAnnotations*

Zuvor wurde schon mehrmals im Zuge dieser Abhandlung der scheinbare Bedarf einer Hilfsapplikation offenbart. In der Diskussion zu den organisatorischen Konventionen von RBMST in Kapitel 7.4.3 wurde die manuelle Konsistenzanalyse als Absicherung gegen unbeabsichtigte Injektion von fehlerhaften Schlüsselwortpragmatiken vorgeschlagen, wobei gerade diese zeitraubende und vor allem ebenfalls auch fehlerbehaftete Überprüfung zumindest in Bezug auf die konventionell vereinbarte Syntax hinfällig wäre. Der semantische Bezug hingegen lässt sich jedoch mittels der manuellen Technik bedingt verifizieren, im anderen Fall nur extrem erschwert.

Ein weiterer Aspekt gilt der zuvor in Kapitel 14.2.1.a diskutierten beabsichtigten Änderung einer zuvor injektierten Schlüsselwortpragmatik beziehungsweise auch einer etwaigen Notifikation wissensbezogener Kontexte. Es wäre natürlich, abhängig von der tatsächlichen IT-Integration denkbar, eine exklusive Möglichkeit über das Eingabewerkzeug zur Metadateninjektion zu offerieren und somit ein nachträgliches Abändern entweder nur mittels Zugriffssicherung auf Access-Ebene zu gestatten beziehungsweise auch kontrolliert zu protokollieren.

Auch die wie in Kapitel 7.4.1 dargestellte Pseudochecksumme zur eindeutigen Abbildung des Lesezeichens könnte somit automatisch gelöst werden, da das Eingabetool die korrespondierende Revisionsnummer implizit ermittelt, und dementsprechend gleich automatisch als Suffix verwenden könnte.

14.2.2.b RBMSS-Knoten sollten zu benannten Ressourcen werden

Diese Erkenntnis stellt im Wesentlichen einen gehobenen Wunsch nach mehr Usability dar. Es wurden bereits während der Entwicklungsphase von RBMST ähnlich Verbesserungspotentiale identifiziert, die jedoch im Verhältnis zum Aufwand nicht die gewünschten Effekte erzielt hätten. Eine definitiv sinnvolle und zugleich auch ausdruckstarke Lösung würde sich durch die Benennung der ansonsten anonymen Ressource (blank Node) mit zusätzlichem Hyperlinking, zur

[1] http://mercurial.selenic.com/wiki/ConvertExtension

navigatorischen Unterstützung, anbieten. Es wäre somit eine Notation einer benannten RBMSS typisierten Ressource denkbar.

Eine Referenzierung in der Art *RBMSS:key/ref*, wobei der *key* für eines der Schlüsselworte stünde und *ref* für die Repository Referenz des Triple-Subjekts. Dies wäre auch gemäss der RDF-Spezifikation[1] eine vollkommen zulässige Aussage. Ein weiterer Vorteil würde sich auch durch eine zusätzliche Verlinkung zum konkreten Artefakt ergeben, und somit die derzeitige Lösung über die Verlinkung des Repository Subjekts ersetzen.

14.2.2.c Kantendarstellungsverläufe über Dotsyntax

Eine andere Art der Navigationsverbesserung ließe sich mit optischen Änderungen über die Dot Sprachebene realisieren. Es wurden dazu Kombinationsmöglichkeiten gefunden die die Darstellung signifikant verbessern könnten. Die Möglichkeiten ergeben sich durch eine dynamische Anpassung der Zuweisungen der *overlap* und *splines* Felder während der Erstellung des SVG-Graphen über den Dot Prozessor. Diese Anpassung müsste die Anzahl und die Verschiedenheit der Knoten im Graphen berücksichtigen, um die Ausgabe dementsprechend zu optimieren. Derzeit wird der *platzsparende* Modus mittels *overlap=false* und *splines=true,* wie in der Datei *graph.inf* ersichtlich ist, verwendet. Es gibt aber noch weit mehr Kombinationsmöglichkeiten die in der Dot Dokumentation[2] angeführt sind. Eine Veranschaulichung möglicher Kantendarstellungsverläufe findet sich in Abbildung 14.2.

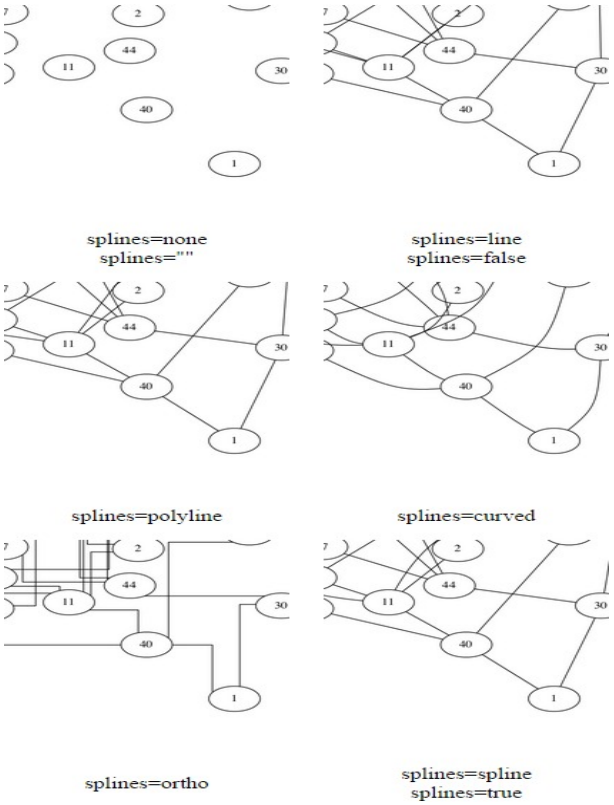


Abbildung 14.2 – Darstellungsvielfalt durch Parameterkombinationen

[1] http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/#ref-rfc-2732

[2] http://www.graphviz.org/doc/info/attrs.html#d:splines

Eine vorherige Festlegung zu definierender Farbverläufe pro Kante würde schon teilweise erhebliche Verbesserungen in der Visualisierung leisten können. Eine Kante stellt im Kontext von RDF prinzipiell eine attributierte Relation dar.

Es werden somit Eigenschaften ausgedrückt, die, wenn auch nicht unbedingt zwingend, in kollaborativer Weise mittels einer Farbzuordung (Farbcodes) vereinbaren und mittels dem Dot-Schlüsselwort *color* pro Kante übergeben werden könnten. So wäre zum Beispiel die Kante vcard:name beziehungsweise vcard:email grün, dcterms:type rot und dcterms:date blau um nur mit einigen Beispielen der Vorstellung Kraft zu verleihen. Diese chromatische Aufgliederung ist auch dann nur sinnvoll anzuwenden, sofern die Ausgabe nicht monochrom beziehungsweise in Graustufen erfolgen würde. Mit Hilfe dieser optischen Trennung würden sich in einfacher Art und Weise diverse Eigenschaften des Graphen ad hoc eruieren lassen.

14.2.2.d Optionale Auswahl verschiedener Ausgabedetails

Die Umsetzung eines *Pseudowasserzeichens* mittels Name des Erstellers, der Station sowie des Erstellungsdatums sind prinzipiell erwünscht, da sie eine gewisse Verbindlichkeit ausdrücken. Gleichzeitig dazu entwickelten sich jedoch Ideen wie die Erstellung eines kompletten Graphen über ein Repositorium für Präsentationszwecke oder zum Austausch mit Unternehmen die in kooperativer Weise verbunden sind. In diesen Belangen wäre eine Auszeichnung über das Wasserzeichen aber eher unerwünscht oder optional.

All diese Anregungen bedingen eine feingranulare Steuerung der Ausgabedetails. Dieses paradigmatische Beispiel, wie auch andere, verlangen somit nach einer dedizierten Möglichkeit zur Selektion verschiedener Ausgabedetails. Es wurde dabei die Steuerung des Wasserzeichens, die zusätzliche Einblendung eines firmenspezifischen Logos wie auch die Möglichkeit eines kleinen Plankopfes mit einer kurzen Legende, man denke diesbezüglich nur an die Kostenmetrik, diskutiert und als sinnvoll erachtet.

14.3 Grenzen der Studie

Im Zuge der Thesenbehandlung zeichneten sich, vor allem in der Erprobungsphase, bereits einige Verbesserungspotenziale ab. Um nach Möglichkeit auch Verbesserungen erzielen zu können, müssen jene Limitationen zuvor identifiziert sowie auch klassifiziert werden. Durch die selbstreflektierende Analyse, der potenziell erschlossenen Mängel, konnte prinzipiell in drei Kategorien unterschieden werden.

14.3.1 Annahmen und Abgrenzungen

Die Annahme, dass die Thesenimplementation im Bereich der Versionskontrollverwaltung ihre Nutzbarkeit fand, wurde durch die Hypothesenbewertungen mehrheitlich bekräftigt. Die Restriktion der Annahme durch Abgrenzungen hinsichtlich Projektcharakteristiken und Benutzergruppen, haben sich jedoch nicht als zwingend notwendig erwiesen.

Im Verlauf der kollaborativen Diskursen hat sich vermehrt gezeigt, dass sehr wohl Projekte die nicht *softwarelastig* sind, wie zum Beispiel ein CAD-Dateienmanagement, auch als Informationsbasis einer Metadateninjektion dienen kann. Im Zuge dessen, muss auch die potenzielle Benutzergruppe nicht lediglich auf Techniker und Manager beschränkt werden. So wären zum Beispiel auch Datenbanksysteme mit logistischen Dispositionsaufgaben sicherlich für eine graphische Aufbereitung, im Stil eines PERT-Graphen, jedoch basierend auf schlüsselwortpragmatischen Metadateninjektionen, geeignete Kandidaten.

Die kollaborativen Sitzungen die sich, als Rückgrat der These, als essentiell und förderlich erwiesen haben, hatten zusätzliches Potenzial im Hinblick auf eine allgemeinere Problembehandlung gehabt. Es wäre letztendlich eine breitere Zielsetzung im wissenschaftlichen Sinn vorteilhafter gewesen, als eine zu detailliert geführte Implementationsdiskussion die dann, wie die Erprobungsphase auch zeigte, in zwar implementierten jedoch teilweise spärlich genutzten Leistungsmerkmalen endete. Ein Beispiel dazu, wäre die so gut wie nicht genutzte Komprimierung der Ausgabeartefakte. Sinnvoller hingegen wäre eine vertiefende Diskussion hinsichtlich erweiterter Anwendungsmöglichkeiten, wie beispielhaft Anbindungsmöglichkeiten an bereits bestehende Informationssystem, innerhalb eines Unternehmens zu führen gewesen.

14.3.2 Technologieauswahl

Als fraglich haben sich auch bestimmte technologische Auswahlkriterien ergeben. Die Vorgehensweise der Nutzung bestehender Toolchains zu Transformationszwecken war bezüglich des Standes der Technik und der angebotenen Mainstreamtechnologien der Communities zwar sinnvoll, jedoch behindert dieser Anreiz zugleich auch das konkrete Evaluieren unpopulärerer Methoden, die gerade in bestimmten Konstellationen jedoch den klassischen vorzuziehen wären.

Konkret wurde in dieser Implementation der bereits bekannte Pfad der RDF Transformation über XSLT zur Dot Syntax beschritten. Wie aber schon in Kapitel 8.2.4 angedeutet, gibt es auch Alternativtechnologien die gleichermaßen ein ähnliches Abstraktionspotential zur Wissensabbildung aufweisen. Beispielhaft angeführt sei hier XML-Topic Maps, im weiteren Verlauf auch kurz als XTM bezeichnet, als Konkurrenztechnologie zu RDF. Folgendes Zitat soll die Ausdrucksstärke der beiden Technologien veranschaulichen:

The semantic expressive power of Topic Maps is, in many ways, equivalent to that of RDF, but the major differences are that Topic Maps (i) provide a higher level of semantic abstraction (providing a template of topics, associations and occurrences, while RDF only provides a template of two arguments linked by one relationship) and (hence) (ii) allow n-ary relationships (hypergraphs) between any number of nodes, while RDF is limited to triplets.[1]

Die technische Erleichterung die sich, bedingt durch die Generalität von RDF im Verhältnis zu XTM stellt ist, dass es weit mehr Werkzeuge zur Weiterverarbei-

[1] http://en.wikipedia.org/wiki/Topic_Maps

tung gibt als umgekehrt. RDF kann somit als Lock-In Technologie[1] bezeichnet werden, jedoch wäre es dennoch über Umwege möglich gewesen, XTM in RDF zu überführen[2]. Da der Fokus der Arbeit in der methodischen Thesenbehandlung lag und die Implementation ein bloßes Mittel zum Zweck war, wurden diesbezüglich weitere Nachforschungen eingestellt.

14.3.3 Modus Operandi

Die Art und Weise der Ausführung der Querschnittstudie stellt sicherlich das größte Verbesserungspotenzial dar. Es wurden bereits während der qualitativen Datenerfassung sowie in der post Analyse der Datenauswertung einige Defizite vakant.

So waren teilweise Fragestellungen wie zum Beispiel MF17, TF7, MI7 & MI8 entweder zu allgemein gehalten, oder aber auch, durch eine verzerrte Erwartungshaltung, zu wenig neutral formuliert, um danach objektive Maßstäbe in ihrer Bewertung ansetzen zu können. Der Bewertungsmaßstab, der arithmetische Mittelwert beziehungsweise die relative Mehrheit, sind zwar gängige Faktoren, könnten aber mit zusätzlichen deskriptiven Größen abgesichert werden.

Es wurde letztendlich auch ein leichtes Ungleichgewicht, zwischen konkreten Fragestellungen und Fragen die eine mögliche Weiterentwicklung des Werkzeuges ergründen, erkannt. Somit wurden etwa zu viele vage Fragestellungen hinsichtlich künftiger Planung angestellt, ohne ein danach aussagekräftiges Ergebnis ableiten zu können.

Wie in Tabelle 14.1 dargestellt, wurden nach dem Testzeitraum Probanden stichprobenartig nach dem verbleibenden Eindruck des verwendeten Werkzeuges und der Art und Weise der gestellten Frageformulierungen an sich befragt. Zusammenfassend ergab sich folgender Tenor, der hier paraphrasiert wiedergegeben wird:

„Das Werkzeug hat wirklich Potential um in der Praxis Verwendung zu finden. Bei der Erhebung wurden jedoch auch Fragen gestellt, die nicht immer eindeutig zu interpretieren waren, oder aufgrund des gewählten Testzeitraumes zu verfrüht gestellt wurden.“

Antwortcharakteristik	*eindeutig*	*mehrdeutig*	*offen*
Managementfragebogen	21	2	0
Technikerfragebogen	20	2	0
Managementinterview	9	0	2
Technikerinterview	7	2	1

Tabelle 14.1 – Antwortcharakterisierung der Hypothesen

[1] http://en.wikipedia.org/wiki/Lock-in_%28decision-making%29
[2] Nikita Ogievetsky (2001): XML Topic Maps through RDF glasses

Des Weiteren ist die Aussagekraft einer Querschnittstudie für dieses Szenario generell zu hinterfragen. Gerade Fragestellungen, wie unternehmensinterne Auswirkungen oder betriebswirtschaftliche Effekte, lassen sich bezogen auf den Zeitraum der Studie, wenn überhaupt, nur schwer beantworten.

Eine vergrößerte Kohorte würde die Aussagekraft ebenfalls signifikant erhöhen und auch statistische Standardtests erlauben. Entweder ist somit der projektierte Testzeitraum und/oder die Kohortengröße zu klein, und somit suboptimal dimensioniert worden um in *allen* geforderten Teilaspekten zuverlässige Aussagen treffen zu können.

14.4 Ausblick

Die wissenschaftliche Abhandlung schließt mit der Bekanntgabe der notwendigen Information um den Quellcode von RBMST herunterzuladen und gibt auch einen Ausblick auf zukünftige Anpassungen, wie sie im vorangegangen Kapitel angeschnitten wurden oder aber auch für andere Implementierungsformen die dieses Modell zwar als Basis verwenden, jedoch in kanonischer Form es weiterentwickeln und gegebenenfalls auf andere Systeme übertragen möchten.

14.4.1 Bezugsquelle der Forschungssoftware

RBMST ist ein quelloffenes Projekt welches in der Version 1.0.0 bei dieser Arbeit zu Forschungszwecken eingesetzt wurde. Der gesamte Quellcode wird auf *BitBucket*, einer der größten web-basierten Hosting-Dienste gehalten, und steht derzeit via

hg clone https://bitbucket.org/MarcelloPresulli/RBMST

für Mercurial VCS als *clone* bereit. Die Software selbst wurde unter die 3-Klausel Lizenz von BSD gestellt und ist somit kompatibel zum GPL Lizenzmodell und auch kommerziell verwertbar ohne der Obliegenheit den geänderten Quellcode wieder einpflegen zu müssen. Nichts desto trotz wäre es sehr wünschenswert wenn es dennoch rege Entwicklungstätigkeiten parallel zu denen des Maintainers gäbe.

14.4.2 Pläne zur Weiterentwicklung der Software

Die im Kapitel 14.2 diskutierten Verbesserungsvorschläge werden vom Maintainer dieses Projektes teilweise weiterverfolgt werden, wobei ein Releaseplan im Sinne einer Roadmap nicht verfasst werden wird. Die nachfolgenden Punkte werden der Reihe nach behandelt:

— Stabilitätsverbesserungen von RBMST

— Optionale Auswahl verschiedener Ausgabedetails

— Kompakte syntaktische Form der Schlüsselwortpragmatiken

Diese Punkte die mir selbst ein Anliegen sind werden priorisiert im Laufe der Zeit abgehandelt werden, wobei sich im Zuge dieser Behandlung auch anderer

Themen ergeben könnten. Der letzte Punkt stellt aus meiner Sicht noch einen zusätzlichen Diskussionsbedarf dar der, verstärkt in Richtung allgemeine beziehungsweise offene Ausrichtung, hinsichtlich einer künftigen Generalisierbarkeit geführt werden sollte.

14.4.3 Empfohlene Schritte der nächsten Iterationsstufe

Um die in dieser Arbeit gewonnen Erkenntnisse weiter zu festigen, sollte die nächste Iterationsstufe sinngemäß in Form einer Folgestudie, entwickelt werden. Es sollte, zumindest wie in Tabelle 14.1 ersichtlichen Verteilung, unklarer und ungetätigter Antworten, nachgegangen werden. Des Weiteren ist es auch sicherlich zielführend einige, beziehungsweise alle der folgenden angeführten Maßnahmen zu ergreifen.

— Eine Kohortenvergrößerung von derzeit 16 Personen auf mindestens 32 ist anzuraten, da die statistische Signifikanz stark zunimmt

— Bewertung mittels statistischer Stichprobentests beziehungsweise Quantile

— Eine Verlängerung des Testzeitraumes in Richtung Langzeitstudie, von derzeit 4 Wochen auf ungefähr 12 Wochen

— Im Fall einer Langzeitstudie etwaige abteilungsübergreifende Prozessveränderungen erfassen

—Quantitative Bewertung ökonomischer Effekte & qualitative Nutzwerterfassung

— Eine Neukonzeption, anstatt umfangreicher Programmfunktionalität mehr an unternehmensweiter Systemschnittstellen

—Verstärkung auf den Aspekt des „Tool als Präventivsystem drohender Projektverzüge“

14.5 Schlusswort

Wie sich generell im Verlauf dieses Werkes abgezeichnet hat, ist die Idee oder das Konzept das tragende Element eines jeglichen Fortschrittes. Welche konkrete Form diese Konzeption schlussendlich annehmen wird, hängt von vielerlei Faktoren ab. Trifft die Idee einen Bedarf, sei er wirtschaftlich gebunden oder auch freier sozialer Natur, kann sich wie am Beispiel der Initialimplementation von Linux zu sehen war, daraus eine Eigendynamik entwickeln die unvorhersehbar ist. Es gibt auch Forschungsergebnisse die lange Zeit ad acta liegen und erst Jahre später Einzug halten, bis hin zu jenen, bevor das Modell zur Anwendung kommt, auch schon wieder überholt sind. Letztendlich haben vitale open source Produkte eines gemein – die Stärke einer tatkräftigen *Community*. Diese Instanz kann jedoch nur dann erfolgreich und schlagkräftig operieren, wenn sie zusammenarbeitet.

Kollaboration ist somit jenes Rückgrat auf dem die gesamte Dissertation, von der Genesis der ursprünglichen Ideenvorstellung über die Irrwege der Umsetzung bis hin zu jener hier vorliegenden Implementation, basiert.

Anhang

Die Fragebögen

Management zentrierte Fragen

Version 2.1.0

universität wien

Fragebogen für den RBMST-Projektmanager

Diese Befragung dient zur Erhebung der generellen Nutzbarkeit von RBMST, und wird völlig ANONYM behandelt.
Es werden die Rohdaten statistisch verarbeitet und die Resultate dem Unternehmen auf Anfrage zur Verfügung gestellt.

Die Fragestellungen sind prägnant formuliert, und werden vorab in der Kohorte einzeln erläutert.
Die Fragen sind bewusst unkategorisiert, und nur nach der Art der Auswahlfelder sortiert.
Falls zu einer Frage keine Antwort gegeben werden KANN, bitte das Auswahlfeld leer lassen.

1. Wie viele projektrelevante Meetings mit Entscheidungscharakter wurden in der Erprobungsphase abgehalten ?

   _____Meetings

2. Wie viele Artefaktausgaben wurden während der Erprobungsphase erzeugt ?

   _____Artefaktausgaben

3. Wie oft wurden dabei die graphische Artefaktausgabe herangezogen ?

   _____Mal

4. Hat sich eine bedingt erhöhte Informationsverfügbarkeit eingestellt ?

   ☐ sehr ☐ eher ☐ wenig ☐ nicht

5. Wie oft war während der Informationsbeschaffung das betreffende Repository nicht verfügbar, wenn nicht warum ?

   ☐ 0 ☐ <=3 ☐ <=5 ☐ >5 ☐

6. Wie oft wurde die kryptographische Übermittlung verwendet, wenn ja für welchen Zweck (Vertraulichkeit, Authentifikation) ?

   ☐ Nie ☐ <=5 ☐ <=10 ☐ >10 ☐

7. Hat sich die bedarfsorientierte Informationsintegrität durch das Pseudowasserzeichen erhöht ?

   ☐ sehr ☐ eher ☐ wenig ☐ nicht

8. Wurde der bedarfsorientierte Informationsaustausch bzw. Wissensaustausch durch den Einsatz des Systems forciert ?

   ☐ sehr ☐ eher ☐ wenig ☐ nicht

9. Dienen die Darstellungen einer verbesserten Repräsentation der Information und des Projektzustandes ?

   ☐ sehr ☐ eher ☐ wenig ☐ nicht

10. Ist die Aussagekraft eines Projektzustandes bezüglich der erhöhten Informationstransparenz verbessert ?

    ☐ sehr ☐ eher ☐ wenig ☐ nicht

11. Deckt die Anzahl der vereinbarten Schlüsselwörter, und somit die Ausdrucksform, die Bedarfsfälle ausreichend ab ?

    ☐ sehr ☐ eher ☐ wenig ☐ nicht

Ing. Mag. Marcello Presulli 2014

Version 2.1.0

Fragebogen für den RBMST-Projektmanager Seite 2

12. Ergibt sich durch die Nutzung der Repository-Metadaten ein besseres Gefühl der Sicherheit aufgrund der originären Informationsquelle ?

☐ sehr ☐ eher ☐ wenig ☐ nicht

13. Wurde Zeit bei der Beschaffung relevanter Informationen im Vergleich zur ursprünglichen Erfassungsmethode eingespart ?

☐ sehr ☐ eher ☐ wenig ☐ nicht

14. Ist die Systemverarbeitungszeit, unabhängig von der Repositoryanbindung, für eine adäquate Informationsbeschaffung ausreichend ?

☐ sehr ☐ eher ☐ wenig ☐ nicht

15. Wird das Sicherheitsgefühl bei Entscheidungsfindungen basierend auf den Ergebnissen des Systems bestärkt ?

☐ sehr ☐ eher ☐ wenig ☐ nicht

16. Wurde durch die automatische Erhebung gleichgearteter Informationsbedürfnisse die Ermittlungsphase zeitlich entlastet ?

☐ sehr ☐ eher ☐ wenig ☐ nicht

17. Hat die Anzahl der tatsächlich benötigten Informationsträger abgenommen ?

☐ sehr ☐ eher ☐ wenig ☐ nicht

18. Hat sich insgesamt ein kollektiv verbessertes Projektverständnis eingestellt ?

☐ sehr ☐ eher ☐ wenig ☐ nicht

19. Welche Kommunikationsform wurde zur Beantwortung von Anfragen (enquiry) verwendet, wenn andere, welche ?

☐ email ☐ verbal ☐ andere

20. Zeichnen sich durch die effizientere Informationsermittlung auch künftige Projektengpässe besser ab ?

☐ ja ☐ nein

21. Ist der Detail- und Navigationsgrad der graphischen Repräsentation ausreichend, wenn nein warum ?

☐ ja ☐ nein

22. Hat sich durch die formale Aufbereitung der Information ein konsensualer gruppenübergreifender Projektdiskurs ergeben, wenn ja hauptsächlich wodurch ?

☐ nein ☐ ja

23. Wenn die Artefaktausgabe auch als Entscheidungsgrundlage fungiert hat, welcher Metrikfaktor wurde dabei verwendet ?

☐ nein ☐ ja

Ich bedanke mich vorallem in meinen Namen, und im Namen der Universität Wien für die tatkräftige Unterstützung seitens der Privatwirtschaft.

Ing. Mag. Marcello Presulli 2014

Abbildung 15.1 – Fragebogen für den RBMST-Projektmanager

Technisch orientierte Fragen

Version 2.1.0

universität wien

Fragebogen für den RBMST-Projekttechniker

Diese Befragung dient zur Erhebung der generellen Nutzbarkeit von RBMST, und wird völlig ANONYM behandelt.
Es werden die Rohdaten statistisch verarbeitet und die Resultate dem Unternehmen auf Anfrage zur Verfügung gestellt.

Die Fragestellungen sind prägnant formuliert, und werden vorab in der Kohorte einzeln erläutert.
Die Fragen sind bewusst unkategorisiert, und nur nach der Art der Auswahlfelder sortiert.
Falls zu einer Frage keine Antwort gegeben werden KANN, bitte das Auswahlfeld leer lassen.

1. Wie oft wurde eine konventionelle Konsistenzanalyse der injektierten Schlüsselwortpragmatiken vollzogen ?

   _____Mal

2. Wie viele Fehleingaben bei den Schlüsselwortpragmatiken wurden selbstständig bemerkt und auch korrigiert ?

   ☐ 0 ☐ <=2 ☐ <=5 ☐ >5

3. Welches Schlüsselwort wurden bei der Metadateninjektion am häufigsten verwendet ?

   ☐ state ☐ dependency ☐ enquiry ☐ knowledgeSource

4. Wie lange dauert durchschnittlich die exklusive Eingabe einer Schlüsselwortpragmatik, wenn >90 Sekunden warum ?

   ☐ <=90 Sekunden ☐ >90 Sekunden

5. Sind die voreingestellten regulären Ausdrücke der Filter (\w+) für eine Informationsabbildung und nachträgliche Informationsextraktion ausreichend, wenn nicht welcher reguläre Ausdruck wäre adäquat ?

   ☐ sehr ☐ eher ☐ wenig ☐ nicht

6. Ist die Notationsform der Schlüsselwortpragmatik (Syntax) intuitiv anwendbar ?

   ☐ sehr ☐ eher ☐ wenig ☐ nicht

7. Wie aufwändig ist die Metadateninjektion im Bezug auf die subjektive Nutzbarkeit ?

   ☐ sehr ☐ eher ☐ wenig ☐ nicht

8. Hat sich ein besseres semantisches Verständnis zwischen den Projektbeteiligten etablieren können ?

   ☐ sehr ☐ eher ☐ wenig ☐ nicht

9. Ist die Auflösung der Arbeitspakete durch die Untergliederung der Punktnotation hinreichend ?

   ☐ sehr ☐ eher ☐ wenig ☐ nicht

10. Ergibt sich durch die Nutzung der Repository-Metadaten eine besseres Gefühl hinsichtlich der Informationstransparenz bei Projektentscheidungen ?

    ☐ sehr ☐ eher ☐ wenig ☐ nicht

11. Deckt die Anzahl der vereinbarten Schlüsselwörter, und somit die Ausdrucksform, die Bedarfsfälle ausreichend ab ?

    ☐ sehr ☐ eher ☐ wenig ☐ nicht

Ing. Mag. Marcello Presulli 2014

Version 2.1.0

Fragebogen für den RBMST-Projekttechniker Seite 2

12. Hat sich durch die Nutzung der Ausgabeartefakte auch die projektbezogene Informationsrepräsentation in den jeweiligen Meetings verbessert ?

☐ sehr ☐ eher ☐ wenig ☐ nicht

13. Wäre ein explizites Werkzeug zur geführten Metadateninjektion, welche alle systemischen Anforderungen abdeckt, sinnvoll ?

☐ sehr ☐ eher ☐ wenig ☐ nicht

14. Hat sich durch die Möglichkeit der Metadateninjektion während des Projektverlaufes die Situation bezüglich einer ad hoc Informationsbereitstellung entspannt, wenn ja inwiefern ?

☐ sehr ☐ eher ☐ wenig ☐ nicht

15. Ist eine nachträgliche Modifikation bereits injektierter Schlüsselwortpragmatiken komplex ?

☐ ja ☐ nein

16. Wird die Eingabe sofort nach erfolgreichem commit bewerkstelligt ?

☐ ja ☐ nein

17. Wird die Schlüsselwortpragmatik vom Autor des commits selbst erstellt ?

☐ ja ☐ nein

18. Ist die Eingabe der Lesezeichen über die graphische Oberfläche TortoiseHG komfortabel ?

☐ ja ☐ nein

19. Wird der RBMST-Systemeinsatz oft nur an die Releaseversionen gekoppelt ?

☐ nein ☐ ja

20. Wird die Darstellung des Datums und der Kontaktdaten des Autors in der Artefaktausgabe als unangenehm erachtet ?

☐ ja ☐ nein

21. Wäre eine Definition einer eigenständigen Ontologie (Begriffswelt) von Vorteil ?

☐ ja ☐ nein

22. Sind durch die Bereitstellung der Metadaten auch die gewisse Projektentscheidungen nachvollziehbarer geworden ?

☐ ja ☐ nein

Ich bedanke mich vorallem in meinen Namen, und im Namen der Universität Wien für die tatkräftige Unterstützung seitens der Privatwirtschaft.

Ing. Mag. Marcello Presulli 2014

Abbildung 15.2 – Fragebogen für den RBMST-Projekttechniker

Die Interviewfragen

Management zentrierte Interviews

Version 1.2.0

universität wien

Interviewfragen für den RBMST-Projektmanager

Diese Befragung dient zur Erhebung der Verwendbarkeit von RBMST, und wird völlig ANONYM erhoben.
Die Interviews dienen dazu um subjektive Intentionen und Interpretationen qualitativ besser erfassen und dementsprechend auch effizienter auswerten zu können.

Die Fragestellungen sind bewusst prägnant formuliert, und werden während des Interviews im Bedarfsfall erläutert.
Falls zu einer Frage keine Antwort gegeben werden KANN, wird der Vermerk K.A. gesetzt und nicht weiter bewertet.

1. Hat sich durch die Workflowabbildung auch ein nachvollziehbareres Projektbild ergeben ?

2. Welche projektspezifischen Abfrageeinstellungen sind am sinnvollsten und warum ?

3. Sind interimistische Abfragen zwecks genauerer Projektstandsfeststellung sinnvoll ?

4. Wird das Konfliktmanagement durch den Einsatz des Systems beeinflusst ?

5. Erzeugt das System eine negatives Gefühl hinsichtlich der Projekttransparenz ?

6. Sind durch die Nutzung des Systems verstärkt gruppeninterne Konflikte entstanden ?

7. Haben sich durch die Systembenutzung auch Kommunikationskanäle zwecks Informationsbeschaffung verändert ?

Ing. Mag. Marcello Presulli 2014

Version 1.2.0

Interviewfragen für den RBMST-Projektmanager Seite 2

8. Könnte das System andere, auch persönliche, Informationsschnittstellen verdrängen ?

9. Entstehen durch die Benutzung des Systems auch psychosoziale Effekte (Stress,Erleichterung), wenn ja welche ?

10. Bringt das System auf überregional verteilte Entwicklungen einen Mehrgewinn ?

11. Wäre die Definition einer eigenständigen Begriffswelt (Ontologie) von Vorteil ?

Die hier notierten Zusammenfassungen vermitteln meine getätigten Intentionen völlig korrekt.

firmenmäßige Zeichnung

Ich bedanke mich vorallem in meinen Namen, und im Namen der Universität Wien für die tatkräftige Unterstützung seitens der Privatwirtschaft.

Ing. Mag. Marcello Presulli 2014

Abbildung 15.3 – Interviewfragen für den RBMST-Projektmanager

Technisch orientierte Interviews

Version 1.2.0

universität wien

Interviewfragen für den RBMST-Projekttechniker

Diese Befragung dient zur Erhebung der Verwendbarkeit von RBMST, und wird völlig ANONYM erhoben.
Die Interviews dienen dazu um subjektive Intentionen und Interpretationen qualitativ besser erfassen und dementsprechend auch effizienter auswerten zu können.

Die Fragestellungen sind bewusst prägnant formuliert, und werden während des Interviews im Bedarfsfall erläutert.
Falls zu einer Frage keine Antwort gegeben werden KANN, wird der Vermerk K.A. gesetzt und nicht weiter bewertet.

1. Hat sich eine bedingte Workflowabbildung negativ ausgewirkt, wenn ja warum ?

2. Wurden korrekte Schlüsselwortpramatiken bewusst nachträglich entfernt ?

3. Gibt es interne Konsequenzen bei Nichtbefolgung der Konventionen ?

4. Welcher funktionale Bereich hat die Konventionsanalyse vollzogen, und wie ?

5. Wurde das Wissensmanagement durch den Einsatz des Systems beeinflusst ?

6. Erzeugt das System eine negatives Gefühl hinsichtlich der Projekttransparenz ?

7. Sind durch die Nutzung des Systems verstärkt gruppeninterne Konflikte entstanden ?

Ing. Mag. Marcello Presulli 2014

Version 1.2.0

Interviewfragen für den RBMST-Projekttechniker Seite 2

8. Könnte das System andere, auch persönliche, Informationsschnittstellen verdrängen ?

9. Entstehen durch die Benutzung des Systems auch psychosoziale Effekte (Stress, Erleichterung), wenn ja welche ?

10. Bringt das System auf überregional verteilte Entwicklungen einen Mehrgewinn ?

Die hier notierten Zusammenfassungen vermitteln meine getätigten Intentionen völlig korrekt.

firmenmäßige Zeichnung

Ich bedanke mich vorallem in meinen Namen, und im Namen der Universität Wien für die tatkräftige Unterstützung seitens der Privatwirtschaft.

Ing. Mag. Marcello Presulli 2014

Abbildung 15.4 – Interviewfragen für den RBMST-Projekttechniker

Glossar und Abkürzungsverzeichnis

Canvas	eine begrenzte digitale Zeichenfläche
Collection	Scala-Standardbibliothek zur Containerhaltung von Daten
Commit	Vorgang des Eincheckens eines Changesets in das Repository
Changeset	logische Menge aller veränderten Daten seit letzem commit
Holistisch	ein synonymer Begriff für ganzheitlich
Mechatronik	Verbinden mechanischer und elektrischer Komponenten
Mockup	Wegwerfprototyp
Performance	allgemeiner Begriff für den Leistungsaspekt/Steigerung
Projektsnapshot	Terminus einer Projektabschnittsfestlegung
Scrum	agiles Vorgehensmodell das mittels Sprints bewältigt wird
AES	Advanced Encryption Standard
API	Application Programming Interface
COCOMO	Constructive Cost Model
CRC	Cyclic Redundancy Check
DSL	Domain Specific Language
GUI	Graphical User Interface
IRI	Internationalized Resource Identifier
JVM	Java Virtual Machine
NFS	Network File System
PERT	Program Evaluation and Review Technique
RBMSS	Repository Based Management Support System
RBMST	Repository Based Management Support Tool
SBT	Simple Build Tool
SCM	Software Configuration Management
UNC	Uniform Naming Convention
UTF	Universal Character Set Transformation Format
VCS	Version Control System

Quellen- und Literaturverzeichnis

[AA2004] Aschauer Anneliese et al. (2004)
Psychosoziale Gesundheit am Arbeitsplatz S. 20 66

[AB2010] Aiella Bob et al. (2010)
Configuration Management Best Practices 23

[AD2007] Abadi Daniel et al. (2007)
Scalable Semantic Web Data Management Using Vertical Partitioning. VLDB 2007, S 411–422 78

[AG1971] Akerlof George (1971)
The Market for Lemons 47

[AG2009] Anderson Gail et al. (2009)
Essential JavaFX, Kapitel 5 152

[AJ2009] Armstrong Joe (2009)
Programming Erlang, Pragmatic Programmer Press 120

[BB2002] Boehm Barry (2002)
Get ready for agile methods, with care.Seiten 64-69 125

[BC2007] Badica Costin (2007)
Advances in Intelligent and Distributed Computing: Proceedings of the 1st Symposium of IDC 133

[BH2008] Balzert Helmut (2008)
Lehrbuch der Softwaretechnik, Springer Verlag S. 151 ff. 34

[BJ2001] Bruno Jenny (2001)
Projektmanagement in der Wirtschaftsinformatik, S. 451 46

[BJ2008] Bode Jeanette (2008)
Performance Measurement und Management, Igel Verlag, S. 3 ff. 168

[BJ2009] Bosch Jan 2009
From Software Product Lines to Software Ecosystems 41

[BK2000] Beck Kent (2000)
eXtreme programming explained 126

[BM2013] Broy Manfred (2013)
Projektorganisation und Mangement im Software Engineering 37

[BR2002] Böhm Rolf (2002)
System-Entwicklung in der Wirtschaftsinformatik, S. 206 ff. 145

[BS2005] Baerisch Stefan (2005)
Versionskontrollsysteme in der Softwareentwicklung 26

[ČI2000] Čeh Ines (2000)
Ontology Driven Development of Domain-Specific Languages 74

[CO2000] Corcho Oscar (2000)
A Roadmap to Ontology Specification Languages 73

[DF2008] Dimmel Friedrich (2008)
Usability Enhanced Agile Development for Web based Applications, S. 53 126

[DI2007] Dhotre I.A. (2007)
Operating Systems, Pune Publication, S. 5 - 34 105

[DJ2000] Dittmann Jana (2000)
Digitale Wasserzeichen: Grundlagen, Verfahren, Anwendungsgebiete; Xpert Press 142

[DJ2008] Johnson Donald (2008)
A Scalability Study of Web-Native Information Visualization, Mississippi State University 137

[DN2000] Degele Nina (2000)
Informiertes Wissen: eine Wissenssoziologie der computerisierten Gesellschaft, S. 96 208

[DN2002] Degele Nina (2002)
Einführung in die Techniksoziologie 11

[DT2003] de Marco Tom (2003)
Bärentango: Mit Risikomanagement Projekte zum Erfolg führen 37

[EB2009] Brigitte Eller (2009)
Usability Engineering in der Anwendungsentwicklung S. 177 126

[EO2007] Erling Orri et al. (2007)
RDF Support in the Virtuoso DBMS. S 59 – 68 77

[FJ2003] Frost John (2003)
Learn SVG: The Web Graphics Standard 139

[FJ2009] Friedl Jeffrey (2009)
Reguläre Ausdrücke, O'Reilly, S. 69 ff. 132

[FM2010] Fowler Martin (2010)
Domain Specific Languages, Addison-Wesley, Chapter 7 75

[FN2014] Norman Fenton (2014)
Software Metrics: A Rigorous and Practical Approach, Francis Verlag 147

[GD2006] Daniel Galorath et al. (2006)
Software Sizing, Estimation, and Risk Management, Auerbach Publications, S. 32

[GE2010] Gemma Edwards (2010)
Mixed-Method Approaches to Social Network Analysis 32

[GL1999] Gräf Lorenz (1999)
Online Research: Methoden, Anwendungen und Ergebnisse S.161 168

[GM2012] Gupta Munish (2012)
Akka Essentials, Packt Publishing 121

[GR2005] Gonzalez Roberto (2005)
A Semantic Web approach to Digital Rights Management, University of Barcelona Thesis 118

[HA2001] Henrich Andreas (2001)
Management von Softwareprojekten 36

[HA2012] Heinecke Andreas (2012)
Mensch-Computer-Interaktion: Basiswissen für Entwickler S.19 ff 123

[HD2013] Hinojosa Daniel (2013)
Testing in Scala, O'Reilly, S.9 - 19 120

[HE2006] Harold Elliotte (2006)
Java I/O, O'Reilly, S. 174 143

[HI2014] Hanschke Inge (2014)
Lean IT-Mangement, Hanser Verlag, S. 403 - 406 184

[HJ2009] Hebeler John (2009)
Semantic Web Programming, Wiley Publishing, S. 8 ff. 136

[HM2004] Hetmank Maik (2004)
Open Source Software: Eine ökonomische Betrachtung, Grin Verlag 103

[IETF2014] Internet Engineering Task Force (2014)
RFC 7231 Chapter 6.4.4 82

[IT2014] Icom Team (2014)

Constructive Communication in international Teams 47

[IW2013] ICom Workshop (2013)
More Agility trough Technology, ICom Team Buch S.207 ff. 208

[KC2013] Kostka Claudia (2013)
Der Kontinuierliche Verbesserungsprozess S.42 ff. 93

[KD2004] Doug Kaye (2004)
Loosely Coupled: The Missing Pieces of Web Services S.132 100

[KG2004] Klyne Graham et al. (2004)
Resource Description Framework (RDF): Concepts and Abstract Syntax 77

[KH2005] Krcmar Helmut (2005)
Informationsmanagement S. 280 58

[KJ2012] Kerschbaumer Jörg et al. (2012)
Collaboratively constructed knowledge repositories as a resource for domain independent concept extraction 31

[KM2011] Kofler Michael (2011)
Linux 2011: Debian, Fedora, openSUSE, Ubuntu 108

[KS2010] Koranne Sandeep (2010)
Handbook of Open Source Tools, Springer, S. 435 140

[KT2012] Karle Thomas (2012)
Kollaborative Softwareentwicklung auf Basis serviceorientierter Technologien, Scientific Publishing 70

[LD1994] Donald Lewine (1994)
POSIX Programmers Guide S. 153 ff. 112

[LJ2009] Loeliger Jon (2009)
Versionskontrolle mit Git, O'Reilly Verlag, S. 289 ff. 87

[LK2012a] Louden Kenneth (2012)
Programming Languages, Course Technology, S. 30 - 35 61

[LK2012b] Louden Kenneth et al. (2012)
Programming Languages: Principles and Practices, Course Technology, S. 216 113

[LM2012a] Lang Michael et al. (2012)
Perfektes IT-Projektmanagement: Best Practices, S. 361 ff. 46

[LM2012b] Lewis Mark (2012)
Introduction to the Art of Programming Using Scala, Chapman & Hall, S. 543 147

[LS2009] Losh Steven (2009)
A Guide to Branching in Mercurial 116

[MC2013] Muller Christophe (2013)
Instant Mercurial SCM Essentials How-to, Packt Publishing, Kapitel Working with other SCM's 116

[ME2006] Merker Erwin (2006)
Programmieren lernen mit Java: Leicht verständlich - Griffige Beispiele, Vieweg Verlag, S. 41 ff. 153

[MG2010] Meinhold Günther (2010)
Der Wissenswürfel: Ein Zustandsmodell des Wissens 43

[MH2003] Mayer Horst (2003)
Interview und schriftliche Befragung: Entwicklung, Durchführung und Auswertung, Oldenbourg Verlag, S. 86 ff. 168

[MH2012] Moosbrugger Helfried (2012)
Testtheorie und Fragebogenkonstruktion S. 18 ff. 58

[MM2005] Mernik Marjan (2005)
When and how to develop domain-specific languages. ACM Computing Surveys, Vol. 37, S 316-344 74

[MM2009] Milanović Milan et al. (2009)
Model Transformations to Bridge Concrete and Abstract Syntax of Web Rule Languages. Computer Science and Information Systems, Vol. 6 S 47-85 73

[MN2004] Mahle Niko (2004)
Das technische und strategische Potential von SVG, diplom.de 148

[MR2008] Martin Robert (2008)
Clean Code: A Handbook of Agile Software Craftsmanship 98

[MR2009] Motschnig Renate et al. (2009)
Konstruktive Kommunikation. Sich und andere verstehen durch personenzentrierte Interaktion, Verlag Klett-Cotta, S. 122 ff. 47

[MT2010] Moser Thomas (2010)
Engineering Project Management 54

[NK2012a] Nagel Katja (2012)
Professionelle Projektkommunikation, Linde Verlag, S. 9 45

[NK2012b] Nagel Katja (2012)
Professionelle Projektkommunikation, Linde Verlag, S.15 ff. 46
[NT2004] Nguyen Tien et al. (2004)
Architectural Software Configuration Management in Molhado, IEEE Computer Society Press 29

[OA2010] de Oliveira Arantes et al. (2010)
An Infrastructure for Managing Semantic Documents 30

[ÖB2007] Österreicher Bernd et. al. (2007)
APM - Agiles Projektmanagement: Erfolgreiches Timeboxing für IT-Projekte 37

[OB2009] O 'Sullivan Brian (2009)
Mercurial: The Definitive Guide 87

[OB2014] O'Sullivan Brian (2014)
Mercurial: The Definitve Guide, chapter 6 91

[OM2008] Odersky Martin et al. (2008)
Programming in Scala,Artima Press, S. 251 ff. 154

[PA2002] Paff Anja (2002)
Die Konstruktion von Fragebögen und ihr möglicher Einsatz in der Evaluation, Grin Verlag, S. 5 - 8 169

[PG2008] Patzak Gerold et al. (2008)
Leitfaden zum Management von Projekten und Projektportfolios und projektorientierten Unternehmen 38

[PJ2003] Park Jack (2003)
XML Topic Maps: Creating and Using Topic Maps for the Web S. 330 ff. 118

[PK2013] Passig Kathrin (2013)
Weniger schlecht programmieren, O'Reilly Verlag, S. 307 102

[PM2011] Projekt Management Austria (2011)
Das Standard projekthandbuch 86

[PP2014b] Pausch Peter (2014)
Lean Project Managment, hanser, S. 185 ff. 60

[PP2014a] Pausch Peter et al. (2014)
Lean Project Management: Projekte exzellent umsetzen 49

[PS2003a] Powers Shelley (2003)
Pratical RDF Book Chapter 4.2 79

[PS2003b] Powers Shelly (2003)
Practical RDF, O'Reilly, S.121 80
[RG2004] Robles Gregorio et. Al. (2004)
Remote Analysis and Measurement by Means of the CVSAnalY Tool," 27

[RJ2006] Reeckie John et al. (2006)
A Software Architecture Primer, Angophora Press, S. 11 ff. 126

[RS2004] RDF- Syntax Specification (2004)
RDF/XML Syntax Specification (Revised), Kapitel 2.10 134

[RU2004] Reimer Ulrich (2004)
Von textbasiertem zu inhaltsorientiertem Wissensmanagement, Business Operations Systems 98

[SA2009] Sarma Anita 2009
Interactive Visual Exploration of Socio-Technical Relationships in Software Development 32

[SB1996] Schneier Bruce (1996)
Angewandte Kryptographie - Protokolle, Algorithmen und Sourcecode in C 142

[SE2001] Senst Ernst (2001)
Virtuelle Teamarbeit, Sensed-Media 109

[SE2007] Scherm Ewald 2007
Organisation: Theorie, Gestaltung, Wandel ; mit Aufgaben und Fallstudien 38

[SH2010] Steinmnann Hendrik (2010)
Informationsaustausch zwischen den Unternehmensorganen im Rahmen der Corporate Governance Diskussion 55

[SJ2009] Schütze Jens (2009)
Modellierung von Kommunikationsprozessen in KMU-Netzwerken S.34 ff. 47

[SJ2012] Suereth Joshua (2012)
Scala in Depth, Manning Press, S. 201 ff. 146

[SL2014] Scala Language (2014)
The Scala Language, Section License 120

[SM2007] Skulschus Marcus et al. (2007)
XSLT, XPath und XQuery, S 29. 137

[SM2008] Subbarao Meera (2008)
DZone, http://java.dzone.com/articles/javadoc-or-doxygen 98

[SR2008] Studer Rudolf et al. (1998)
Data & Knowledge engineering: Principles and methods Vol.25, S 161-198 72

[SR2014] Steyer Ralph (2014)
Einführung in JavaFX: Moderne GUIs für RIAs und Java-Applikationen, Springer Verlag, S. 63 ff. 160

[SS2009] Staab Steffen et al. (2009)
Handbook on Ontologies. Springer Verlag 73

[SV2009] Subramaniam Venkat et al. (2009)
Practices of an Agile Developer, Pragmatic Programmer 124

[SV2011] Vedran Sabol (2011)
2.2.3.c Knowledge Extraction and Integration using Automatic and Visual Methods 31

[TD2004] Thomas David et al. (2004)
Versionsverwaltung mit CVS, Hanser 24

[TP2009] Parr Terence (2009)
The Definitive ANTLR Reference: Building Languages 64

[WD2008] Wampler Dean (2008)
Programmieren mit Scala, O'Reilly, S. 393 ff. 154

[WD2009] Wampler Dean (2009)
Programming Scala: Scalability = Functional Programming + Objects, O'Reilly S.369 121

[WD2010] Wielsch Dan et al. (2010)
Governance of Massive Multiauthor Collaboration, University of Berkeley 87

[WJ2012] Weaver James et al. (2012)
Pro JavaFX 2: A Definitive Guide to Rich Clients with Java Technology, APress 122

[WK2005] Wiegers Karl (2005)
Software-Requirements, MS Press, S. 219 ff. 173

[YL2011] Yang Li (2011)
Applied Cryptography for Cyber Security and Defense, University of Tennessee, S. 135 143

[YS2009] Yanchar Stephen et al. (2009)
Beyond the Theory-Practice Split in Instructional Design, 2009, S. 88-89 15

Abbildungsverzeichnis

Abbildung 1.1 – Design Oriented Research 16

Abbildung 2.1 – Mercurial Interaktionsmechanismus 26

Abbildung 2.2 – Extraktion aus diversen Repositorien 28

Abbildung 2.3 – Die Toolchain des Semantic Document Repository 30

Abbildung 2.4 – Graph Visualierung über Tesseract Browser 32

Abbildung 3.1 – Sneed's Teufelsquadrat 35

Abbildung 3.2 – Aufgaben des Software-Projektmanagements 36

Abbildung 3.3 – Der IT-Projektlebenszyklus 37

Abbildung 3.4 – funktionale und produktbezogene Aufteilung 39

Abbildung 3.5 – Der Projektleiter als Schnittstellenfunktion 41

Abbildung 3.6 – ein vielseitiges Projektmanagementtool 43

Abbildung 4.1 – Projektkommunikation und deren Steuerungen 45

Abbildung 4.2 – wichtige Einflussgrößen und deren Kommunikationswirkung 46

Abbildung 4.3 – Varianz zwischen Auslastung und Prozessdauer 48

Abbildung 4.4 – die Todeszone der Aufwandsabschätzung 50

Abbildung 4.5 – Formel des gesteigerten Kommunikationsoverhead 50

Abbildung 4.6 – Optimale Ermittlung der Mitarbeiteranzahl 51

Abbildung 5.1 – DSL versus GPL ein Vergleich 64

Abbildung 6.1 – Repository Based Management Support 70

Abbildung 6.2 – W3C Sprachschichten des Semantic Webs 73

Abbildung 6.3 – Abbildung einer Ontologie auf eine DSL 74

Abbildung 6.4 – graphische Endbenutzerschnittstelle TortoiseHg 77

Abbildung 6.6 – RDF-Schema der RBMSS Schlüsselworte 83

Abbildung 6.7 – RBMST generierter SVG-Graph des RBMSS-Schemas 84

Abbildung 7.1 – Branching & Tagging 90

Abbildung 7.2 – eine injektierte Schlüsselwortpragmatik samt Etikett 91

Abbildung 7.3 – der Metadatensatzaufbau von Mercurial 94

Abbildung 7.4 – die syntaktische Generalform der Schlüsselwortpragmatik 95

Abbildung 7.5 – Kommunikationsparadigma der enquiry Aktion 96

Abbildung 8.1 – die Mercurial PlugIns der Standarddistribution 102

Abbildung 8.2 – das Standardthema der integrierten WebServers 104

Abbildung 8.3 – beispielhafte Kommunikation des Command Servers 106

Abbildung 8.4 – Grundlegende Parametereinstellungen via XML-Tagging 115

Abbildung 8.5 – selbst entwickelte Pseudosyntax zwecks Graphennotation 117

Abbildung 9.1 – Modulblöcke der Software RBMST 126

Abbildung 9.2 – eine vollständige RBMST-Konfigurationsdatei 128

Abbildung 9.3 – Auszug aus den in RBMST verwendbaren Java-Patterns 131

Abbildung 9.4 – anonymer typisierter RDF Knoten aus dem RDF Tutorial 134

Abbildung 9.5 – Relation zwischen Repository-Subjekt und einem RBMSS-Objekt 135

Abbildung 9.6 – sämtliche Ausgabetransformationen von Graphviz Dot 141

Abbildung 9.7 – die par Methode und ihre Verwendung 146

Abbildung 9.8 – die dimensionslose Kostenmetrik 148

Abbildung 9.9 – ein stark vergrösserter Ausschnitt eines Projektsnapshots 149

Abbildung 10.1 – Synopsis von RBMST auf Kommandozeilenebene 155

Abbildung 10.2 – selektives Aus- und Entkommentieren von Projektparametern 159

Abbildung 10.3 – die graphische Bedienoberfläche von RBMST 161

Abbildung 10.4 – ein vollständiger Projektgraph in SVG Serialisation 163

Abbildung 11.1 – der Startbildschirm von Grafstat 169

Abbildung 11.2 – die divisionale Aufteilung der Klinger Gruppe 175

Abbildung 13.1 – Beispielauswertung eines Interviews für den RBMST-Manager 197

Abbildung 13.2 – Beispielauswertung eines Interviews für den RBMST-Techniker 199

Abbildung 14.1 – kompaktere Syntaxform der Schlüsselwortpragmatik 216

Abbildung 14.2 – Darstellungsvielfalt durch Parameterkombinationen 218

Abbildung 15.1 – Fragebogen für den RBMST-Projektmanager 226

Abbildung 15.2 – Fragebogen für den RBMST-Projekttechniker 228

Abbildung 15.3 – Interviewfragen für den RBMST-Projektmanager 230

Abbildung 15.4 – Interviewfragen für den RBMST-Projekttechniker 232

Auflistungsverzeichnis

Auflistung 9.1 – eine Template Datei von RBMSS 130

Auflistung 9.2 – eine Reasoner Regel für anonyme Knoten 136

Auflistung 9.3 – ein XSL-Ausschnitt der Datei rd2dot.xsl 138

Auflistung 9.4 – das exekutierte XSL-Makro zum vorigen Aufruf 139

Auflistung 9.5 – ein Graphenbeispiel in Dot-Syntax 140

Auflistung 9.6 – die Datenstromverkettung von GZip und Cipher 144

Auflistung 9.7 – eine effiziente Datenstromselektion 145

Auflistung 9.8 – die Unvorhersagbarkeit paralleler Abarbeitung 146

Auflistung 9.9 – parallel genutzte Datenstrukturen in RBMST 147

Auflistung 10.1 – die zentrale Builddatei von sbt 153

Auflistung 10.2 – beispielhafte Warnungs- und Fehlerausgaben 154

Auflistung 10.3 – die Parameter des Kommandozeilenaufrufes 155

Tabellenverzeichnis

Tabelle 6.1 – verwendete RDF Eigenschaften und deren Namensräume 81

Tabelle 8.1 – Gegenüberstellung der unterschiedlichen Betriebsarten 112

Tabelle 10.1 – Bedeutung der zentralen Parameter von RBMST 156

Tabelle 10.2 – Bedeutung der Schlüsselpragmatikparameter von RBMST 157

Tabelle 10.3 – Bedeutung der schlüsselwortspezifischen Parameter 158

Tabelle 10.4 – Bedeutung der projektspezifischen Parameter 158

Tabelle 11.1 – Vergleich zwischen Kurz- & Langzeitstudie 167

Tabelle 13.1 – Die Grundauswertung des RBMST-Manager Fragebogens 190

Tabelle 13.2 – Die Grundauswertung des RBMST-Techniker Fragebogens 195

Tabelle 13.3 – Bewertung der Fragebögen des RBMST-Managers 200

Tabelle 13.4 – Bewertung der Fragebögen des RBMST-Technikers 201

Tabelle 13.5 – Bewertung der Interviewfragen des RBMST-Managers 202

Tabelle 13.6 – Bewertung der Interviewfragen des RBMST-Technikers 202

Tabelle 13.7 – Ergebnis der konsolidierten Hypothesenbewertungen 206

Tabelle 14.1 – Gegenüberstellung der synchronen & asynchronen Lösung 214

Tabelle 14.1 – Antwortcharakterisierung der Hypothesen 221

Curriculum Vitae

Persönliche Daten

Name: Marcello Presulli
Geboren: 06.07.1973 in Baden, Österreich
Staatsbürgerschaft: Österreich
Status: verheiratet, eine Tochter geboren 2000

Ausbildung

2009 — 2015	PhD in Wirtschaftsinformatik, Universität Wien
2000 — 2006	Wirtschaftsinformatik, Universität Wien und Technische Universität Wien

Berufliche Laufbahn

2014 —	selbstständig im Bereich Softwareentwicklung und Consulting
2009 — 2013	Mitbegründer der Fa. Embed-IT CEO
2004 — 2009	Keytronix GmbH. Head of Development Embedded Software Engineer
2000 — 2004	Datentechnik GmbH., KeyMile AG Embedded Software Engineer Produktmanager
1998 — 2000	Mitbegründer der Fa. Virtual Business CEO Unix System & Netzwerkadministrator